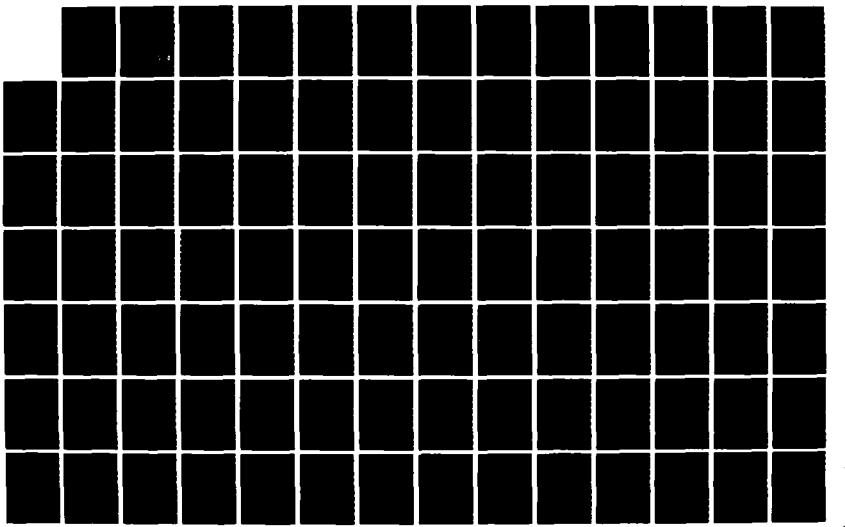
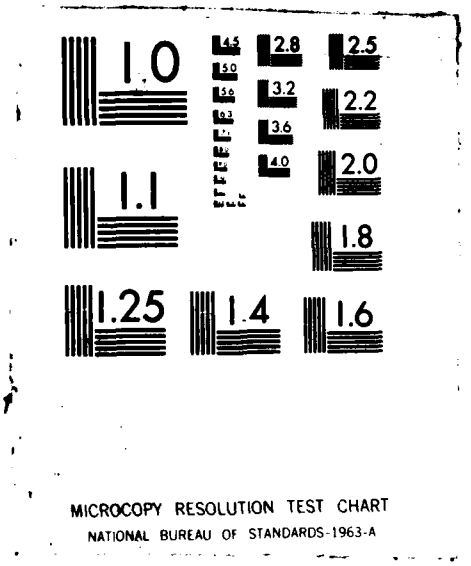
AD-A177 717 PROTOTYPE KNOWLEDGE ACQUISITION AND REPRESENTATION EDITOR FOR THE F-16 FLIGHT DOMAIN(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. 1/2

UNCLASSIFIED D A SOBOTA DEC 86 AFIT/GCE/ENG/86D-4 F/G 9/2 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD-A177 717


# PROTOTYPE KNOWLEDGE ACQUISITION AND REPRESENTATION EDITOR FOR THE F-16 FLIGHT DOMAIN

THESIS

Darleen Avery Sobota
Captain, USAF

AFIT/GCE/ENG/86D-4

DTIC FILE COPY

DTIC
ELECTE
MAR 1 6 1987
S D
E

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

This document has been approved for public release and sale; its distribution is unlimited.

87 3 12 088

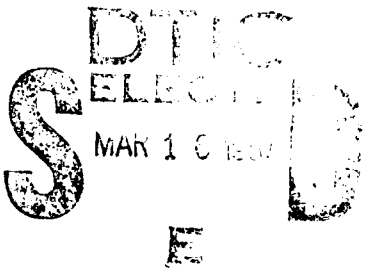
AFIT/GCE/ENG/86D-4

# PROTOTYPE KNOWLEDGE ACQUISITION AND REPRESENTATION EDITOR FOR THE F-16 FLIGHT DOMAIN

THESIS

Darleen Avery Sobota
Captain, USAF

AFIT/GCE/ENG/86D-4




Approved for public release; distribution unlimited

AFIT/GCE/ENG/86D-4

# PROTOTYPE KNOWLEDGE ACQUISITION AND REPRESENTATION EDITOR FOR THE F-16 FLIGHT DOMAIN

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Computer Engineering

QUALITY INSPECTED 4

Accession For
NTIS GRA&I
DTIC TAB
Unannounced
Justification
By
Distribution/
Availability Codes
Dist | Avail and/or Special
A-1

Darleen Avery Sobota, B.S.
Captain, USAF

December 1986

Approved for public release; distribution unlimited

## Preface

The purpose of this research was to build an editor for knowledge acquisition and representation appropriate for a pilot aid in the F-16 flight domain. The loss of lives in fighter aircraft accidents motivates my interest in this project.

I thank my advisor, Major Stephen E. Cross of the Air Force Institute of Technology, Artificial Intelligence Laboratory, for his endless motivation, patience, and realistic guidance. Without him this thesis would not have been possible. I thank my readers Dr. Matthew Kabrisky, Captain David Fautheree, and especially Dr. Freda Stohrer for their technical, organizational, and grammatical critique of this research. I thank my sponsors, the Flight Dynamics Laboratory, Flight Control Division and the Pilot's Associate Program Office, both of the Air Force Wright Aeronautical Laboratories for their computer support and comments on my prototype knowledge editor and final thesis draft, respectively. I also want to thank Major Dick Frank for his F-16 flight domain expertise and enthusiasm, as well as his wife Dr. Betsy Frank for her critique of two of my drafts.

I thank God and the Air Force for sending both my husband, Captain Mark Sobota, and I to school together at the same time. Although in different programs, together we grew intellectually and spiritually on our numerous dates to

the laboratories and computer rooms. The joys and frustrations we shared during this schooling opportunity will long be remembered. Last, but not least, I am truly grateful for the unrelenting support and encouragement provided by my Dad and Mom living in Florida. Their love has made this a growing experience.

Darleen Avery Sobota

## Table of Contents

## List of Figures

## Abstract

Current research in the study of intent inferencing for the fighter aircraft flight domain is hampered by the problem, "How should knowledge required about the flight domain be acquired and represented?" Without automated tools for acquiring and representing knowledge, large expert system development will continue to be hindered.

The purpose of this research was the initial design and construction of a prototype editor to assist in the acquisition and representation of script-based knowledge in the context of the F-16C flight domain. The editor was responsible for: maintaining the internal consistency of the knowledge base when knowledge was added, updated, or deleted; preventing syntax errors in formatting by providing cascading menus for domain specific knowledge entry, while leaving the frame-based formatting details to the editor tool, and; preventing spelling errors among already existing (created) knowledge items by providing pilot selectable pop-up menus reflecting the current state of valid choices in the knowledge base for a particular item. Other potential benefits of an automated knowledge acquisition and representation editor are discussed.

As designed and implemented, the editor generates an internal frame-based representation of the pilot's air-to-ground weapon delivery checklist information, and further, generates a LISP computer program from this internal representation, without requiring the pilot to know the

computer language.

The prototype editor was implemented on a Symbolics 3670 LISP machine, in LISP, supported by Intellicorp's Knowledge Engineering Environment (KEE) frame-based expert system building tool. In the future, integrating plan-based knowledge into the knowledge editor tool should allow testing of the concept of intent inferencing, supporting the work of the Pilot's Associate Office.

# PROTOTYPE KNOWLEDGE ACQUISITION AND REPRESENTATION EDITOR FOR THE F-16 FLIGHT DOMAIN

## I. Introduction

### Background

Imagine a pilot flying the Air Force's most advanced tactical fighter aircraft, a machine with voice controlled input, and computer generated graphical display feedback. As he flies into combat, bombs blast all around him. The number of tasks he needs to perform far exceeds the number humanly possible. The pilot needs an intelligent assistant that "understands" the support required, without the pilot explaining his desires to the assistant.

Part of the solution to the above situation is contained in the concept of intent inferencing. The New World Dictionary defines "intent" as "one's purpose, will, and determination at the time of doing an act" (36:733). "Inferencing," on the other hand, is "the deriving of a conclusion in logic by either induction or deduction" (36:721). In this paper, "intent inferencing" refers to a method of determining a pilot's goals based on the actions he performs (16:160).

The problem of information overload, usually occurring at critical times in a mission, has led to the development of artificial intelligence (AI) techniques to provide an intelligent aid to the pilot. The Air Force Studies Board

supports the AI approach for reducing the burden on the pilot at the man-machine interface (16:160). As a result of this study, the Pilot's Associate (PA) Program Office was founded in the Air Force Wright Aeronautical Laboratories (AFWAL) at Wright Patterson Air Force Base, Ohio. This project is one of DARPA's (Defense Advanced Research Projects Agency) three initiatives under the Strategic Computing Program (SCP) announced in 1983. The real focus behind the Pilot Associate's program is assistance during combat, by increasing survivability and effectiveness. It is not oriented toward a peacetime benign interaction (34).

Loss of airplanes and human life have seriously concerned the Air Force. Trained investigators determine whether the cause was a faulty aircraft or pilot error. The Air Force Inspection and Safety Center has published projections for 1986 military aircraft, the lowest rate ever, which state that 62 aircraft would be lost, with 68.4% due to pilot error. In addition, the center estimated that 66.6% of the aircraft destroyed due to pilot error would be of the fighter and the attack categories (41:6).

One of the subgoals of the Pilot's Associate Program Office is to combat aircraft losses by providing an intelligent computerized aid to the pilot, which assists in decision-making while flying. However, correctly allocating tasks to the pilot or to the intelligent computer aid will be possible only if the machine has information about the goals of the operator. "Intent inferencing provides a

mechanism for the machine to compile information about the intentions of the human operator without the need for detailed explanatory input from the operator" (16:160).

One of the possible outcomes of intent inferencing is a pilot associate capable of first determining the 'plan' that the pilot is executing, and then monitoring his actions to ensure proper execution of the steps within the selected plan. If the pilot deviates from his plan, and no other plan can be found to justify the action, then the aid should ask the pilot to verify his action (23:I-2). Unfortunately, this type of aid has the potential for increasing system overload.

A second type of pilot aid advises or suggests that a particular action be performed. For instance, the computer may suggest that the pilot change his heading, or decrease his airspeed. It can also inform the pilot that an action requested earlier in the mission has been completed (11:154).

The third level of pilot assistance provides decision aiding as would an aircraft back-seater, such as a flight test engineer. Although the computer capabilities may be limited in certain areas, the pilot is aware of these limitations through training. "The back-seater knows how to perform certain functions, but the pilot provides the guidance or sets the thresholds on when these functions are to be performed and how the back-seater is to interface with the pilot" (11:159). At this third level, the pilot could

program the computer himself, indicating when he wants an action to occur and what feedback he desires. Such an aid conserves the pilot's critical and costly knowledge-based level processing, since the pilot knows and understands what actions the computer will take and when it will take them (11:159).

A fourth type of pilot assistant initiates its own tasking based on an assessment of the surrounding environment. "It can be asserted that the computer understands the pilot if it can produce knowledge that predicts (and if requested, explains) the actions the pilot will take to achieve a goal" (11:160). At this highest level, the pilot and computer work in such harmony that the computer is ready with a feasible solution when the pilot needs the assistance. Unlike the third level where the computer functions more as a backseater, this level provides assistance equivalent to an instructor pilot (11:152,160).

## Problem

Current research in the study of intent inferencing for the fighter aircraft flight domain is hampered by two primary problems, "How should knowledge required about the flight domain be acquired and represented?" and "What should be done when there is no existing script or plan to match the pilot's actions?"

This thesis research addressed the first problem only, "How should knowledge required about the flight domain be acquired and represented?" The problem is generic to all

domains, but will be specifically addressed in the context of the flight domain. Without automated tools for acquiring and representing knowledge, large expert system development will continue to be hindered.

The purpose of this research was the initial design and construction of a knowledge editor that would allow pilot elucidation and input of his own knowledge into the Pilot Associate -- that is, the capability to "brief" his assistant on how things are to be accomplished. Although both script and plan knowledge will be necessary to support the level three pilot aid discussed earlier, only script-based knowledge was modelled in this editor. Definitions and examples of plans and scripts will be presented in Chapter II, "Summary of Current Knowledge".

The second problem, "What should and can be done when there is no existing script or plan to match the pilot's actions?" will probably require the creation of a new plan based on both current and future goals, as well as the utilization of both script and plan knowledge. It appears that solving the second problem depends on solving the first.

There are four potential benefits of an automated knowledge acquisition and representation editor. First, the pilot will be 'writing' computer programs without knowing the target computer language. Second, the knowledge engineering dialogue with the pilot can be replaced by direct pilot usage of the tool, if the complexity of the

internal frame representation of a script is kept hidden from the pilot. Third, if a different implementation language or format is required, the pilot's knowledge need not be re-entered. All that is needed is an additional output language code generator. Fourth, the executable computer programs produced could be transferred onto a plug-in cartridge into an aircraft, thus tailoring it for each pilot (9).

Scope

The difficulties in representing script and plan knowledge are well known. Mr. Norman Geddes, a PhD candidate at the Georgia Institute of Technology, has discussed these problems in the context of the flight domain (16; 17). In addition, Mehdi Harandi, a professor at the University of Illinois at Urbana-Champaign stated that "it is becoming increasingly apparent that the process of rule acquisition is the major bottleneck in the development of expert systems" (20:237). Consequently, this thesis effort focused on the building of an editor to assist in the acquisition and representation of script-based knowledge in the context of the F-16C flight domain. In particular, the air-to-ground weapons delivery mode, a subset of possible checklists was chosen for the test cases. Each aircraft checklist can be thought of as a script, where a "script" is a self-contained package of knowledge about performing some action.

The scope of this research was to provide an editor allowing the user to enter the flight domain information through the use of pop-up, mouse-sensitive menus. The information had to be formatted by the editor, thus providing no possibility of syntactic error. Additionally, spelling errors had to be removed by providing the user with menus of defined scripts and objects from which an item was selected. If the user wished to rename an item on a pop-up menu, the editor had to search through the current knowledge base finding all affected script checklist steps, updating them with the new name, and informing the user of the changed steps. Deletion of items had to be performed in a similar method, thus eliminating inconsistencies in the knowledge base. A method for adding new information about the particular aircraft's flight domain to the appropriate pop-up menu when entered by the user had to be provided, allowing for the continual expansion of domain specific information.

The editor could provide other useful products to the pilot. For instance, a LISP or Ada program generated from the pilot's pseudo-English explanation of a checklist procedure would be valuable. As designed and implemented, the editor generates the internal frame-based representation of the pilot's checklist information, and further, generates a LISP computer program from this internal representation, without requiring the pilot to know the computer language.

Furthermore, a graphical display of the existing

checklists and their component names were provided to aid in understanding and debugging the complex interactions within and among the knowledge base information. The desired final product was a useful tool for an F-16 pilot to represent his understanding of a checklist to a computer. Automating the internal details of this script-based knowledge acquisition process will hopefully provide a foundation for a similar plan-based knowledge editor, as recommended by Knode (23). In the future, integrating the plan-based knowledge into the same computer software tool should allow testing of the concept of intent inferencing.

Assumptions

Four assumptions were made in approaching the knowledge acquisition and script representation problem.

1. The reader has an introductory level understanding of artificial intelligence (AI) concepts, expert systems, and object-oriented programming.

2. The reader is familiar with the basic aircraft terminology, and appreciates the complexity of a fighter pilot's task.

3. The aircraft parameter settings, such as heading, airspeed, and fuel-remaining, are predefined. The editor displays a pop-up menu to the user showing the available parameters for usage while writing constraints.

4. The aircraft coordinate system was defined using a

conventional body axis system (such that pitching up represents a positive value, and pitching down represents a negative value).

Approach

The author performed the following major steps in accomplishing the research:

1. The author performed a literature search into the theory of plans, scripts, and goals to derive an adequate script representation format. The most helpful "approaches to planning" research was provided by Robert Wilensky, Roger Schank, and Harold Abelson. Wilensky investigated the backward chaining process for logically connecting rules to build a plan justifying an action, using the foundation provided by Schank and Abelson. Knode provided a conceptual application of Wilensky's work to the flight domain. Geddes, a previous A-7D pilot, is actually applying all of the above research to the flight domain. Each of these efforts, and other related knowledge acquisition and knowledge representation efforts, will be discussed in Chapter II.

2. An F-16C checklist for the air-to-ground weapon delivery mode provided the conceptual design baseline for this thesis research. Furthermore, several interviews with Major Dick Frank, an F-16 pilot assigned to the F-16 System Program Office (SPO) at Wright-Patterson Air Force Base, supplied the "between-the-line" checklist steps

used by a pilot. Obtaining this knowledge from a domain expert is referred to as knowledge engineering. The knowledge engineering process will be discussed further in Chapter II. Combining the information from the F-16 checklists and the interviews reflected Major Frank's understanding and training, both critical to proper implementation of the checklists. Additional F-16 references included the F-16C Technical Order Manual and the Avionic System Manual, Block 25B (12; 19).

3. An evaluation of the checklists for three test cases chosen for the conceptual and detailed design phases (Chapters 3 and 4, respectively) from the F-16 air-to-ground weapons delivery mode provided a list of requirements a knowledge editor should perform to assist a user in building a script-based knowledge system for the flight domain. Automating the knowledge acquisition process required the process of software engineering, to be defined in Chapter II. Researching automated editing tools uncovered the works of Glenn Abrett and Mark Burstein, Daniel Shapiro and Brian McCune, Medhi Harandi, and Richard Waters. Each of these efforts will also be discussed in Chapter II.

4. An analysis of three expert system building tools: KEE (Knowledge Engineering Environment), ART (Automatic Reasoning Tool), and M.1 was performed to choose the most appropriate tool upon which to build the knowledge

editor. KEE was selected as the expert system building tool since it provided a graphical display of the interaction of the objects, many built-in reasoning options, and a menu-driven environment. Its graphical display of the knowledge was of great benefit in developing the scripts.

5. Chapter III, the conceptual design, contains the preliminary features of a knowledge acquisition and script-based representation editor, resulting from the research discussed in Chapter II.

6. Chapter IV, the detailed design, contains the necessary features of the knowledge editor, resulting from the rapid-prototyped implementation on a Symbolics LISP Machine for the three selected test cases. The prototype demonstrated proof of concept of the knowledge editor. The detailed design can be used to write the specifications for a fully developed and integrated editor in the future should the Air Force decide to fund such an effort.

7. All recommendations for enhancing the user interface, arising out of the prototype implementation, were incorporated into the editor. As expected, use of graphics aided the user in debugging and visualizing the script interactions. A description of the prototype editor is provided in Chapter V.

8. Chapter V concludes with a subjective analysis of the rapid-prototyped knowledge editor by Maj. Richard Frank (F-16 pilot), Lt. Col. Ronald Morishige (F-4 pilot), Maj. Ronald Van der Weert (F-15 pilot), and Norman Geddes (A-7D pilot).

9. Finally, the thesis conclusion and recommendations for the future research are located in the final chapter, Chapter VI.

The approach taken in this research is evolutionary in nature. The pilot assumes the responsibility of representing his understanding of a checklist script accurately, while the computer only uses the information supplied to it. Georgia Tech, on the other hand, is taking the revolutionary approach. Their approach presents understanding as the computer's responsiblity (17).

## Materials and Equipment

A LISP (LISt Processing) Machine was needed for implementation of the rule editor, due to symbolic representation of the data, the built-in procedures for object-oriented programming, the use of KEE as the expert system building tool, and the large memory requirements. The Symbolics 3670, located in the Flight Dynamics Laboratory, Flight Control Division, Control Systems Development Branch (AFWAL/FIGL), at Wright-Patterson Air Force Base, Ohio provided the computer support for this research.

## II. Summary of Current Knowledge

### Pilot's Associate Program

The research performed in this thesis was sponsored by the Pilot's Associate (PA) Program Office located in the Air Force Wright Aeronautical Laboratories (AFWAL) at Wright Patterson Air Force Base, Ohio.

#### Pilot's Associate Overview.

The Pilot's Associate project is one of DARPA's (Defense Advanced Research Projects Agency) original three application initiatives under the Strategic Computing Program (SCP) announced in 1983. Pilot's Associate addresses an Air Force related problem. The other two demonstration areas for the Navy and Army, respectively, are Battle Management Systems and the Autonomous Land Vehicle (2:4). Two new applications are Air Land Battle and "Brilliant" Weapons (22).

The Strategic Computing Program hopes to challenge the research arena so that major breakthroughs in artificial intelligence, computer architecture, and microelectronics for military systems will occur. The AI areas of particular interest are vision, speech, natural language processing, and expert systems technology (2:4).

#### Pilot's Associate Objectives.

According to the Statement of Work for the Pilot's Associate Program, the PA program has two objectives:

1) to demonstrate and evaluate the potential

benefits of artificial intelligence technology for enhancing the mission effectiveness of future combat aircraft, and
2) to provide an application for the emerging technologies of the Strategic Computing Program and other DoD programs. (2:3)

According to Lt. Col. Ronald Morishige, Air Force Program Director for the Pilot's Associate project and an F-4 command pilot, as well as Lt. Col. John Retelle, DARPA Program Manager for the Pilot's Associate project and a graduate of the French Test Pilot's School as a flight-test engineer, the purpose of the PA is to maintain a high level of situational awareness for the pilot by providing him with the information needed at the proper time, instead of just bombarding him with large quantities of random data "The Pilot Associate will integrate, prioritize, filter, and communicate the most significant information in accordance with the current situation" (25:92). The data format must be easily comprehended by the pilot. The pilot can override the PA if he disagrees with a recommendation, or he may request additional information. Yet, if the pilot's perception of the surrounding environment, both in and out of the cockpit is not accurate, the resulting misperception can mean both mission failure and death (26:1-2; 25:92).

Pilot's Associate System.

The functional construct of the Pilot's Associate system is shown in Figure 1. The five critical functions requiring expert advice are system status, situation assessment, tactics, mission planning, and the pilot-vehicle

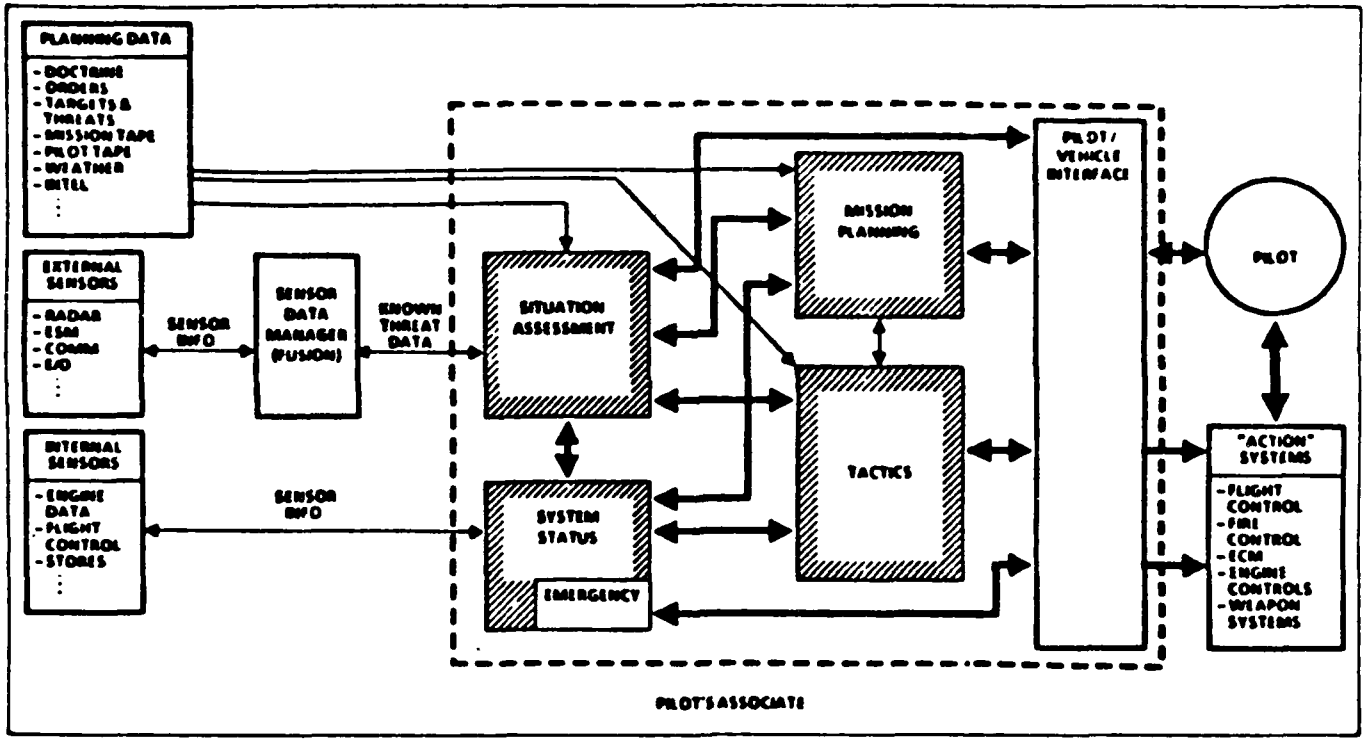



Figure 1. Pilot Associate Block Diagram (31:69; 2:9)

interface. A high level of interaction among these components will be required (31:69). Each block will be briefly discussed.

The systems status box will be responsible for monitoring the aircraft subsystems, such as engines and hydraulics. If a malfunction occurs, this system would provide the pilot with recommended actions, and may even reconfigure the aircraft (26:3).

The situation assessment function is not as well-understood as the systems status box. Functionally, it is the complement of the systems status box; instead of monitoring the internal aircraft state, it must determine the external aircraft state (9). "The success of the entire program could hinge on substantial progress in this single function" (26:4). As an example of the importance of this function, if the PA inaccurately represents the current situation, an excellent solution to the wrong problem will be obtained, while the real problem continues to exist (26:4).

"Tactics are the responsibility of the pilot" (26:5). The PA should not propose the tactic, but instead help the pilot to implement his own tactic with a minimal amount of interfacing (9). However, if the situation assessment function is not properly working, neither will the tactics, wings, flaps, engines, radar, or any other aircraft system. The real challenge lies in providing the PA with the appropriate information so that it can understand the

intentions of the pilot (26:5).

The fourth Pilot Associate function to be addressed is mission planning. Its primary objective is to "provide the pilot with increased flexibility for executing the assigned mission" (26:3). There are many components of mission planning, such as navigating, route changing, and avoiding threats while still having enough fuel to return home safely (26:3).

The last of the primary PA functions is the pilot-vehicle interface. This function is critical because of its direct interaction with the pilot. Care must be taken in the method of presenting the information. "While the pilot-vehicle interface must use the most natural means of communication to transfer information including speech, visual, and tactile modes," each pilot will have his own preference at a particular time in the mission (26:6).

The knowledge acquisition editor developed in this thesis addresses the systems status and pilot-vehicle interface interrelationship. A future thesis effort could build upon this editor, addressing the mission planning function of the PA system.

Pilot's Associate Program Structure.

The PA effort is planned for five years with two phases (26:8). The time-frame for integrating the PA technologies into the combat aircraft is post-1995. However, the initial testing has already begun. Initial testing is being

conducted in simulations, with a final demonstration in a "full-mission, piloted, real-time simulator" (2:3).

Dr. Bill Rouse, President of Search Technology and a professor at the Georgia Institute of Technology, envisions intent inferencing as a crucial component in the future pilot-vehicle interface. Search Technology, is a subcontractor to Lockheed-Georgia, to whom the Pilot's Associate Program Office awarded one of their research contracts (9).

"Inference engine," terminology used by the artificial intelligence community, refers to the section of computer software providing the control mechanism. Another term for inference engine is the "type of reasoning." Search Technology is currently using an inference engine based on Dr. Robert Wilensky's approach in the Plan Applier Mechanism (PAM) and Roger Schank's conceptual dependency (CD) knowledge representation format (28). Yet other knowledge representations and inference engines have not been ruled out totally.

Pilot's Associate Summary.

In conclusion, the PA effort represents exploratory development and several levels of pilot decision aids are anticipated. As discussed in Chapter I, these aids ranged from monitoring the aircraft systems to initiating their own tasking based on the current situation. Lt. Col. Morishige summed up the PA program's direction with the following:

> No computer, not even in 2001, is going to eliminate the requirement for manned aircraft. Missions requirements will certainly not become less demanding than they are today. Performance demands will increase. The best overall performance will be obtained from a synergistic combination of man and machine capabilities. The Pilot's Associate will help the pilot maintain good situational awareness to improve mission effectiveness and survivability. (26:9)

## Introduction to Scripts

According to Wilensky, "the notion of a script was introduced as a way of structuring knowledge about stereotyped situations" (38:6). Scripts represent experiences that occur repeatedly by modelling the performed actions (9). The knowledge editor developed in this research provides a tool for assisting a user in acquiring and representing script-based knowledge. The checklist in an F-16 is an ideal example of a script.

Unfortunately, scripts have limitations to their rigid structure. For instance, if an action occurs that is not present in the script, then it cannot be accounted for in that context (38:6). The typical restaurant script, from the customer's perspective, includes these actions: being seated, ordering from a menu, eating the meal, tipping the waitress, paying the bill, leaving the restaurant. For a computer programmed to recognize this typical script, the unusual behavior of a customer who jumps up and runs out of the restaurant during her meal cannot be explained by the computer. Based on the restaurant script desribed, the computer cannot determine whether the restaurant was on

fire, whether she ran away without paying, or whether she became violently ill and fled.

The limitations of scripts were illustrated in the last example since scripts cannot contain every possible variation. If it were possible to include every variation of a script, doing so would defeat the script concept. The purpose of a script is to constrain the search process by delineating known events found in commonplace context (38:42). When an existing script cannot be used by the computer to justify an action, the action can be explained through combining several plans. Nevertheless, scripts provide a valuable function, and a plan should not be used when an appropriate script is available, since the search process is much more complicated for a plan. The following example will demonstrate the value of a script.

Most artificial intelligence problems involve a tree traversal, using heuristic search techniques. "Breadth" represents the width of the tree, while "depth" represents the height. By definition, a script has a depth of D, equal to the number of steps in the script, and a breadth of 1. Using a script, the search process is very quick, since the appropriate path is established once a script is chosen, thus requiring no backtracking. Therefore, a script provides a time savings in the search process indicated by the relationship $\text{Breadth}^{\text{Depth}}$ ($B^D$). If there are 20 steps in a script, then the number of steps requiring inferencing has been reduced from 20 to 1. As a result, reasonable depth of

comprehension can be gained through the use of scripts (9).

Definition of Plans, Goals, and Constraints

To ensure common terminology thoughout this thesis, three key terms will be defined. 1) A "plan" is an approach selected to accomplish some desired action (38:5). "Plans describe the set of choices that a person has when he sets out to accomplish a goal" (29:70). Once a plan is selected, all the steps in that plan must be accomplished (18). 2) A "goal" is the state resulting from the last step in a successfully executed plan (9). Activating a plan to achieve one or more goals is common in everyday situations. 3) "Constraints" are circumstances that suppress an action. A procedural example demonstrating the interaction of these three terms follows:

A plan for starting a car includes the following steps:

1. Insert key in ignition.
2. Turn key clockwise while pressing accelerator.
3. Hold key in full clockwise position until purring engine is heard.

The goal is to hear the purring engine.

A constraint is to have the correct key. (9)

Introduction to Plans

A plan, which can be thought of as a subcomponent of a script, is an approach selected to accomplish a desired action. Wilensky gives an example of a plan: "A person realizes that a friend's birthday is coming up and decides to go to the drugstore to buy that friend a birthday card"

(38:5). This plan is just one of many plans that could have been chosen. The selection process could be based on previous experiences, on consulted advice, on protocol procedures, or on some other reason. The gift chosen by the giver will probably be equivalent to the gift she received for her birthday from her friend. Planning, thus, includes four areas: 1) assessing the overall situation, 2) deciding what goals should be pursued, 3) creating the plans to secure these goals, and finally, 4) executing the plans (38:5).

## Scripts and Planning Research Reviewed

Several researchers have contributed to progress in the area of scripts and plans. In particular, the research of Schank and Abelson, Wilensky, Faletti, Knode, Geddes, Stefik, Davis, and Feigenbaum will be discussed in more detail to gain a better understanding of the requirements of the knowledge editor.

### Schank and Abelson's Work on SAM and Conceptual Dependencies.

Scripts, plans, and goals as related to story understanding were pursued by Professors Roger Schank and Harold Abelson of Yale University. They said that people have acquired much knowledge about scripts and plans through their everyday endeavors. For instance, if Mike is hungry, then he may select "the go to McDonald's" script. On the other hand, if Mike is trying to impress a woman he invited

out to dinner, then a hamburger at McDonald's will never do; experience has taught him that a more formal environment is necessary. Experience has also taught people to recognize the intent of John as demonstrated in the next story. "John needed money. He got a gun and went to a liquor store" (3:308). Schank and Abelson point out that although we understand that John intends to rob the liquor store with the goal of obtaining money, the story does not specify that a robbery script should be used (3:308).

Schank's and Abelson's work in the field of natural language processing emphasizes the knowledge representation. They created a knowledge structure for understanding stories called conceptual dependency (CD) form (38:6). The eleven primitive actions defined in this language are: ATRANS, PTRANS, MTRANS, PROPEL, MBUILD, ATTEND, SPEAK, GRASP, MOVE, INGEST, and EXPEL (28:17). Primitive actions were also necessary for representing knowledge in the editor, although modelling of the flight domain required its own set of primitive actions such as SET, DEPRESS, and RELEASE. These primitives are discussed in Chapter IV, "Detailed Design."

Schank and Abelson processed each sentence in a story, converting it into their CD format. For instance, each and every sentence had four parts: 1) an ACTOR, 2) an ACTION performed by that actor, 3) an OBJECT that the action is performed upon, and 4) a DIRECTION in which that action is oriented (28:11). In this theory, the four parts are referred to as slots, and the goal is to fill each slot with

a value. This slot-filling approach was later the basis of Minsky's frame theory. This CD layout is illustrated by the statement: Steve gave Darleen the book. In this case,

ACTOR = Steve
ACTION = give
OBJECT = book
DIRECTION = Darleen.

According to CD theory, the action 'give' would be represented by the CD primitive ATRANS, meaning a transfer of possession.

Schank's and Abelson's CD representation language was first tested by a computer program called MARGIE.

> MARGIE was an attempt to demonstrate that language can be mapped into a deep-level, language-independent conceptual base, and then mapped back out into either the same or another language without losing any essential information. (28:4)

CD theory proved adequate as a knowledge representation form; however, additional work testing the area of context was still needed. Without context, irrelevant inferences were obtained in the search process, thus, resulting in a combinatorial explosion (28:5).

SAM (Script Applier Mechanism) extended the work accomplished by MARGIE, but attempted to get rid of immaterial inferences. SAM proved that when the context of a story was added, a computer system could process and understand a script-based story (28:6).

Wilensky's Work on PAM.

Next in the evolution of ideas was PAM (Plan Applier Mechanism). Under the direction of Schank at Yale

University, PAM was developed by Dr. Robert Wilensky to bridge the understanding gap where an appropriate script did not exist. Wilensky is currently a professor at the University of California at Berkeley.

PAM can understand stories with multiple goals, multiple actors, revised goals, and revised plans (38:49). SAM could not handle stories where a new twist arose. PAM, however, made use of goals and plans to handle this new situation. Schank and Abelson pointed out that "much of story understanding involves understanding the intentions of the characters in the text" (28:136).

Wilensky's notion of planning includes: 1) assessing a situation, 2) determining the goals to pursue, 3) finding or building plans to attain these goals, and 4) executing these plans (23). His computer program McPAM (Micro PAM), a scaled down version of PAM, utilizes 4 distinct types of rules. These four are:

1) Instantiation rules - relate events to plans they may be part of;

2) Planfor rules - relate plans to goals to which they may be applicable;

3) Subgoal rules - relate goals to plans to which they may be instrumental in achieving;

4) Initiate rules - relate themes to goals they may give rise to. (28:192)

The interaction among the rules is illustrated in Figure 2.

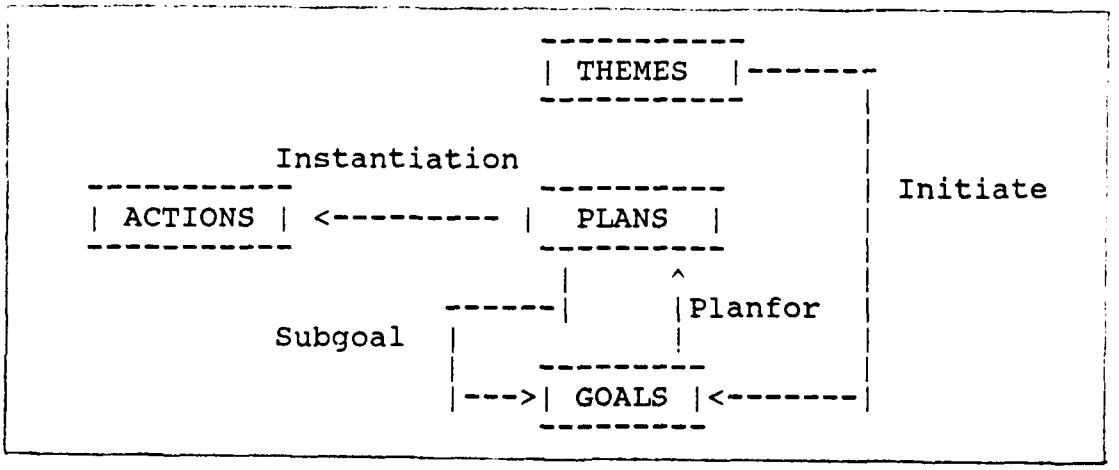



Figure 2. McPAM's Four Rule Types and Interactions

When written in an if-then rule format, they appear as:

Instantiations are :
if plan then action

Planfors are:
if goal then plan

Subgoals are:
if plan then goal, or
if goal then goal

Initiates are:
if theme then goal

When two goals conflict, the complexity of story understanding increases. In this situation, some action must be taken to resolve the conflict. As a result of the conflict, Wilensky researched what knowledge was required about the planning process, arriving at a conflict resolution solution he termed "meta-planning" (37:205). The resolving process is complicated, since there are plans

about plans and goals about goals. Wilensky's meta-planning approach was successfully applied to the autonomous vehicle for mission planning (32).

To demonstrate the meta-planning process, Wilensky provided the following example:

> John's wife called him and told him they were all out of milk. He decided to pick some up on his way home from work. In order to produce a plan, it is necessary to go through the following processes:
>
> 1) Realizing that the goal of getting home and getting some milk are overlapping, and that they should be pursued together rather than independently.
>
> 2) Adjusting one's plans accordingly. In this case, the plan is modified so as to:
>
> a) Produce a route that takes the planner near the grocery store.
> b) The "go home" plan is suspended at the point at which the grocery store is reached.
> c) The "get milk" plan is executed.
> d) The "go home" plan is resumed. (37:205)

To generate the plan above, two meta-knowledge facts were utilized. These two facts are: it is desirable to achieve a goal with minimum effort and two plans with similar actions provide savings when executed together (37:205-206).

Wilensky summed up meta-planning by saying: "meta-goals are a set of goals for the planning process and meta-plans are a set of plans to achieve them" (37:205). In addition to meta-goals and meta-plans, Wilensky organized all his meta-planning activities under what he called meta-themes. His four main strategies were (38:31):

1. Don't waste resources.
2. Achieve as many goals as possible.
3. Maximize the value of the goals achieved.
4. Avoid impossible goals.

One last point stressed by Wilensky was the clear distinction between computer understanding and computer problem solving, although both areas utilize planning information. For problem solving, the computer must create a plan to satisfy a given goal. On the other hand, for understanding, the computer makes inferences only when both the goals and plans of actors are followed. "Rather than actually create a plan, an understander (the computer) must be able to use knowledge about plans to understand the plan under which someone else is operating" (37:199). The intent inferencing portions of the Pilot Associate deals with understanding, not problem solving.

In conclusion, Wilensky's goal-based stories using PAM were more interesting than Schank's and Abelson's script-based stories using SAM. Nevertheless, considering PAM's limitations on the types of stories it, too, can understand and SAM's advantage of limiting the search process, SAM remains a key understanding tool.

Faletti's Work on PANDORA.

Joseph Faletti, a student of Wilensky's at the University of California at Berkeley, developed a computer program known as PANDORA (Plan ANalyzer with Dynamic Organization, Revision, and Application). PANDORA used the ideas of PAM and meta-planning as its foundation. While PAM

manipulated linguistic utterances only, PANDORA performed reasoning by the direct manipulation of representations (38:37). PANDORA includes a plan proposer which arrives at a possible meta-plan for a goal. According to Faletti, "PANDORA detects its own goals in an event-driven fashion, dynamically interleaving the creation, execution, and revision of its plans" (14:185).

PANDORA is best known for its application in the UNIX operating system domain. In this domain, PANDORA acts as the problem-solving unit for the UNIX Consultant (UC), an intelligent natural language interface between the user and UNIX (38:38). Dr. Wilensky, supervisor of the development of the UC, said that unlike most natural language front-ends, the UNIX Consultant is not connected to a data base only, but is a learning aid for the users of the UNIX operating system's environment (39:29).

Knode's Work on Planning in the Flight Domain.

Captain David Knode, an Air Force C-130 pilot, under the direction of Major Stephen Cross (a professor at the Air Force Institute of Technology) researched planning for the flight domain. Specifically, an analysis of Wilensky's PAM was performed to assess its appropriateness and flexibility for modelling actions in the complex flight domain.

Knode concluded that Wilensky's theory of planning combined with Rieger's commonsense algorithms (CSA) would be a likely representation for system testing (23:VI-6). He

stated that Wilensky's planning approch was suited for handling the conflicting goals in the inflight emergency domain because it handled not only a priori goals, but also multiple goals. Previous planning research could not handle the multiple goals. PAM was also found appropriate because it could "detect its own goals, determine the priorities of conflicting goals, and abandon goals which are not possible to achieve" (23:VI-3).

Knode believes the real challenge for this flight domain application will be the representation of planning knowledge in a format which enhances the ease of goal understanding. He gave this example of a simple pilot aid in his thesis entitled An Approach to Planning in the Inflight Emergency Domain:

> If the pilot reduces the power, begins a descent to a lower altitude, and lowers the landing gear, the Pilot Associate (PA) should infer that the pilot intends to land the aircraft. The PA then matches the pilot's actions against those outlined on its stored 'landing procedure plan.' If a match between the pilot's actions and a stored plan is found, the PA continues to monitor the pilot's actions. If no match is found, the PA tells the pilot to reconfirm his actions. (23:I-2)

Knode further believes pilot acceptance from a "back-seat" computer aid will occur gradually, as varying levels of assistance are accepted. The pilot must perceive the computer's intention as increasing his situation awareness, thus resulting in mission success. However, the pilot associate will never replace the pilot (8:220).

Geddes' Work on Intent Inferencing.

Norman Geddes, a PhD candidate at the Georgia Institute of Technology, is investigating the feasibility of intent inferencing applied to the flight domain. Intent inferencing is a procedure which attempts to determine what a pilot is trying to do or should be doing based on his actions and current surrounding environment...."Intentions are not, in themselves, observables; they must be inferred from observable actions" (16:162).

Geddes research uses as its foundation the works of the previously discussed plan and script researchers. His work is partially sponsored by the Pilot's Associate Program Office. Geddes is studying the features to be included in the pilot-vehicle interface. More generally, his work utilizes the adaptive aiding framework for the man-machine interface. "The goal of adaptive aiding is to enhance human operator performance while reducing the consequences of human error" (16:161). Adaptive aiding consists of four components: information management, dynamic task allocation, error classification and remediation, and goal conflict resolution (16:161). Geddes' illustration showing the key ideas considered by a computer in determining what the pilot is trying to do is shown in Figure 3.

In representing pilot intentions, the Geddes inferencer closely follows Wilensky's Plan Applier Mechanism (PAM) as recommended by Knode for the flight domain (16:166). Geddes stated that "an intention is formally represented as an

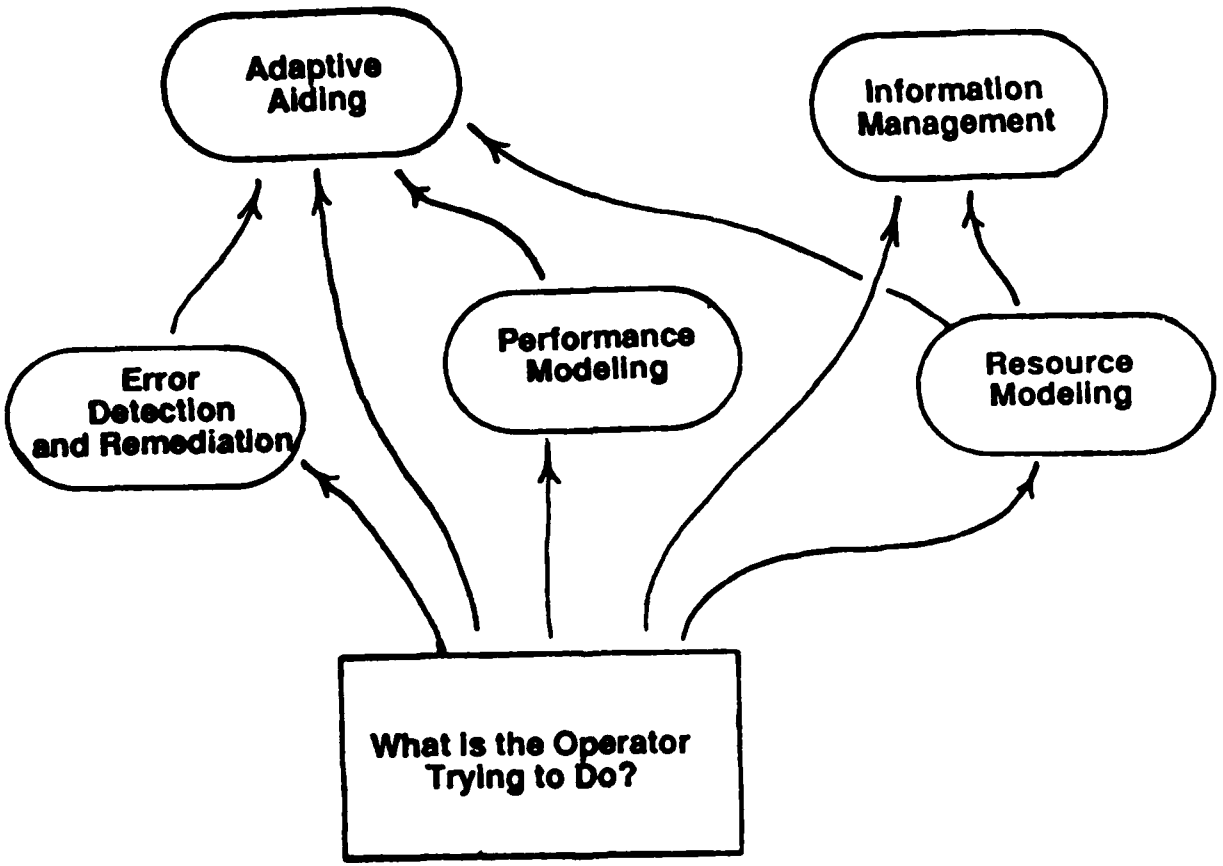



Figure 3. Intent Inferencing Key Ideas (18)

active goal and the particular active plans that are in progress that support the goal" (16:163). Geddes' testing of scripts, plans, and goals in explaining pilot actions is occurring between intercommunicating LISP machines at Lockheed-Georgia's facility. Figure 4 depicts Geddes' model for inferring pilot intentions.

The internal frame representation shared among the computers was derived from Schank's CD format mentioned earlier (16:164). "The scripts and plans representation resembles a case-grammar, with primitive actions and case slots which may be filled by a specific object name or value, or by a variable name" (16:164). The knowledge bases, which will be expanded, currently include information for the ingress flight phase and the beyond-visual-range air combat engagement.

Using Figure 5, the rules for an enemy encounter would be linked together using PAM as shown:

1) Write a rule that indicates that the instantiation of a ready-weapons plan is to select AIM weapons action.
2) Write a rule that relates the subgoal of the ready-weapons plan to the goal of destroying-opponents.
3) Write a rule that initiates the goal of destroying-opponents to the achievement theme. (16:171)

The computer applies these planning rules during flight by first sensing the pilot's actions, then interpreting them, and finally, determining the pilot's goal (what he is trying to accomplish). In the above rules, based on the

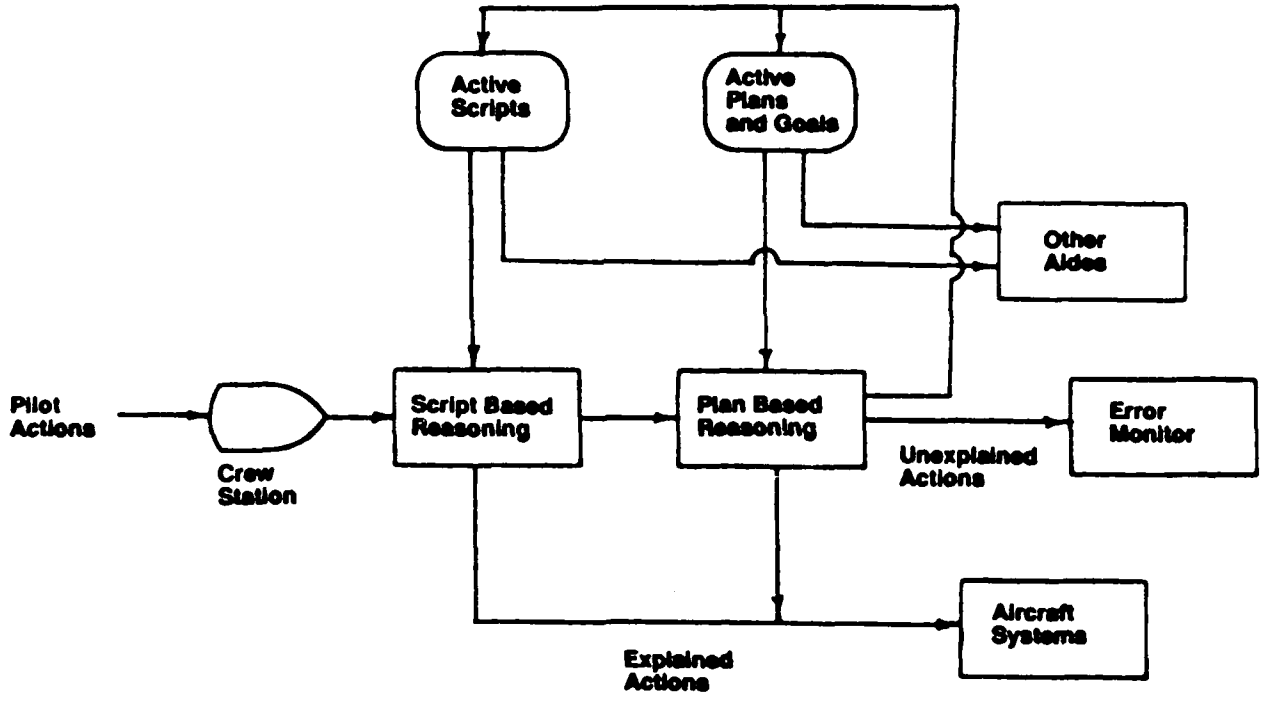



Figure 4. Model for Inferring Pilot Intentions (18)

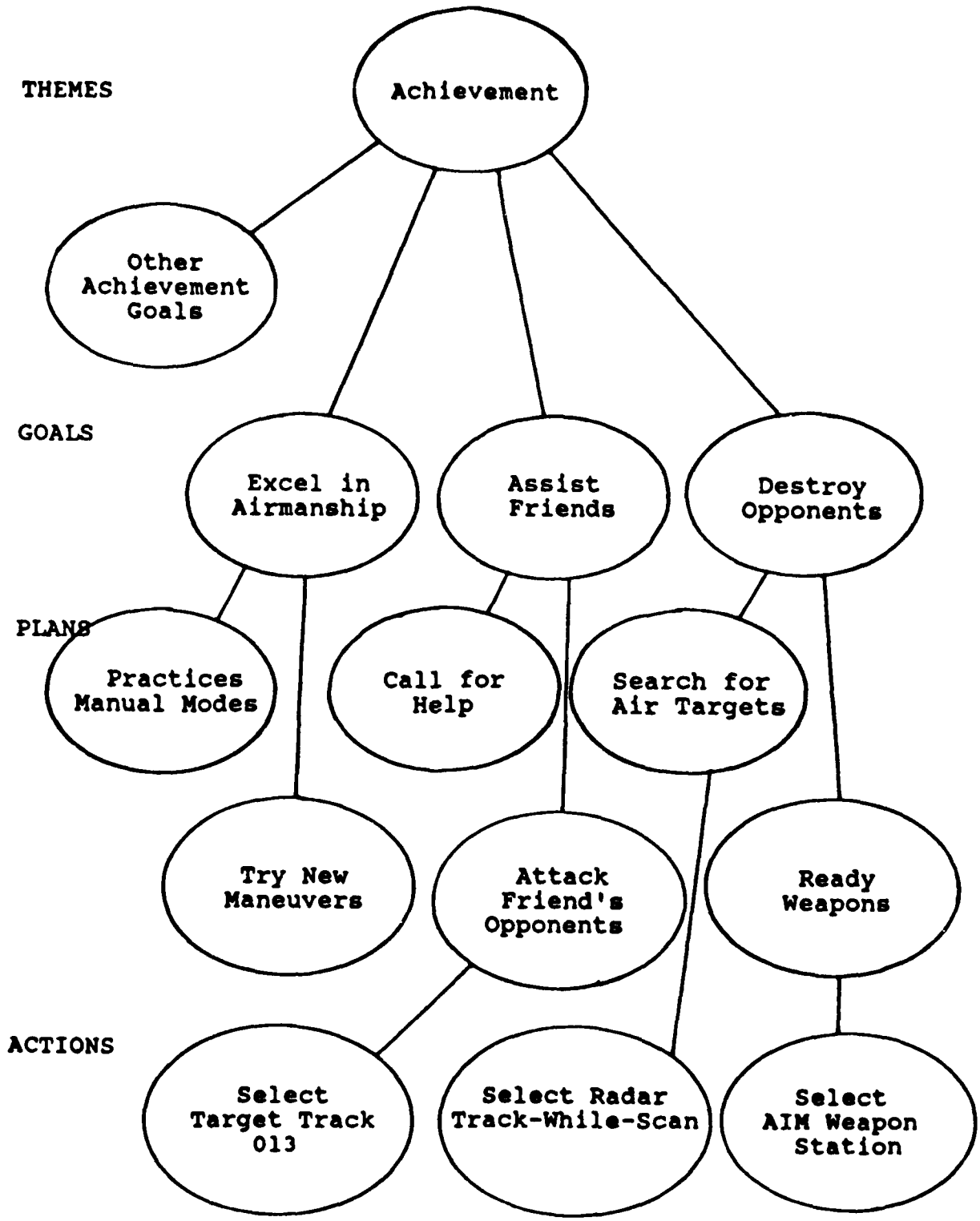



Figure 5. Flight-Domain Theme, Goal, Plan, Action Hierarchy (16)

computer sensing the pilot's selection of AIM weapons, as in rule #1, the ready-weapons plan is activated. The computer then finds a matching goal in rule #2 that ties in the goal of destroying-opponents. Finally, this goal is related to the achievement theme through rule #3. This process becomes very computer intensive as the number of plans used to achieve a goal increases. However, if the above plan were stereotypical, then a supporting script could provide "the mechanism for representing the correct sequencing of both required and optional steps of a procedure, allowing for detection of omitted steps or incorrect ordering" (16:163).

Although the flight applications of Geddes' research have yet to be fully simulated, understanding a pilot's intentions offers many possible levels of automation for the future fighter cockpit. His prototype computer program has exhibited good performance in its small mission environment (16:167).

Stefik's Work on MOLGEN.

Mark Stefik further extended Wilensky's meta-planning ideas into an area termed hierarchical planning. Hierarchical planning is a technique used to solve difficult problems, by hiding the proble' 'etails, until a solution addressing the main issues has been found. This type of planning uses multiple levels of abstraction to develop entire plans. On the other hand, nonhierarchical planning orders operations at only one level of abstraction (7:531). His MOLGEN (MOLecular GENetics) expert system has three

layers of problem-solving spaces, each addressing problems uncovered at the previous layer space (38:39-40). Stefik applied meta-planning to control any issues surfacing during problem-solving using its least-commitment strategy. "This strategy dictates that it defer decisions for which it lacks constraints, and thus, it rarely commits itself to a decision that it must later undo" (7:556).

Davis' Work on TEIRESIAS.

Addressing the rule representation problem, meta-knowledge was applied in Randall Davis' system, TEIRESIAS. In this system "ordinary knowledge is expressed in rules, and knowledge about how to use this knowledge is expressed in meta-rules" (38:38-39). The meta-rules took responsibility for the sequence of other rules, by making predictions about ordinary rules (4:57).

TEIRESIAS is best known for its acquisition of domain specific knowledge for MYCIN, a medical consultation system for infectious diseases, developed at Stanford University (4:87). TEIRESIAS provided an interactive user interface to the human expert so he can monitor the knowledge-based system's performance (4:84,101).

> TEIRESIAS's goal is to reduce the role of human intermediary in this task of knowledge acquisition, by assisting in the construction and modification of the system's knowledge base. (4:87)

Davis extended Wilensky's meta-planning work one step further by exploring the use of meta-knowledge, which he

defined as "the representation in the program of knowledge about the program itself--about how much it knows and how it reasons" (4:89). The meta-level knowledge represents the internal computer program world, while the object-level knowledge represents the external world. Davis' research addressed the problem of appropriately representing and using knowledge (4:89). This problem continues to exist, and therefore, is the research for this thesis.

Feigenbaum's Work on DENDRAL.

Edward Feigenbaum, a well-known AI figurehead, is a professor at Stanford University. Feigenbaum suggested in 1977 that:

> The painful process of knowledge engineering, which involves domain experts and computer scientists working together to design and construct the domain knowledge base, is the principal bottleneck in the development of expert systems. (4:84)

He worked on the DENDRAL program which finds possible molecular structures for some unknown molecule, given spectroscopic analysis (4:106). Due to the combinatorial explosion occurring though exhaustive problem-solving techniques, DENDRAL's knowledge was converted into heuristics. The heuristic knowledge, also called rules of thumb, used by expert chemists in solving the molecular structure problem were finally modelled in DENDRAL (4:115).

> DENDRAL was one of the first programs to demonstrate the power of encoding domain-specific, heuristic expertise and was therefore one of the first projects to recognize knowledge acquisition as a major problem in AI. (4:115)

Scripts and Plans Summary

The several key script-based and plan-based research efforts highlighted in the preceding discussion have advanced AI knowledge. Nevertheless, Major Cross believes that applying the results of the efforts to the flight domain requires solving at least three problems: qualitative reasoning, common-sense knowledge representation, and surrounding world comprehension (8:220). Furthermore, Cross envisions the pilot aid to someday provide the following functions: understand mission objectives and plans, access relevant on-board data and algorithms, anticipate decisions the pilot must make, prioritize and shift attention, "see" cockpit displays, play "what-if?," manage on-board resources, simultaneously work multiple problems, and respond to pilot commands (10:54).

The synergistic result of integrating the Script Applier echanism (SAM) and the Plan Applier Mechanism (PAM) into a powerful inference engine is currently being explored for the flight domain. Additionally, knowledge representation formats continue to be tested.

In summary, the problem of knowledge acquisition and representation has prompted the need for automated tools. As a result, the design and implementation of a knowledge editor became the purpose of this thesis. To adequately build such a tool, software engineering and knowledge engineering principles were utilized.

## Introduction to Software Engineering

In designing the F-16 knowledge editor, the following software engineering principles listed in a book by Richard Fairley were considered: correctness, completeness, consistency, unambiguous, functional, verifiable, traceable, and easily modifiable (13:93). The editor was implemented using the rapid prototyping lifecycle approach to software engineering. A prototype is a model of a software product. Unlike a simulation model, a prototype contains real parts of the delivered product (13:49). "Typically, a prototype exhibits limited functional capabilities, low reliability, and/or inefficient performance" (13:49). Nevertheless, the prototype provides the following valuable functions: a better understanding of the user's needs to the developer, a better understanding of the processing capabilities of the product to the user, and a testbed for exploring technical issues prior to committing resources to the total idea (13:49-50). Furthermore, the cost of implementing a prototype is estimated at 10 percent of the system cost. The remaining system specifications are written from the prototype; however, the prototype is typically thrown away afterwards (21).

## Introduction to Knowledge Engineering

According to Teknowledge, the stages of the knowledge engineering projects are essentially the same as software engineering. Unlike software engineering, the system design

and specifications follow the prototype implementation and the users are involved throughout the process (33:6-13). A project must be identifed, assessed, prototyped, expanded, piloted, and finally, fielded (33:4-1). When building a knowledge system, a good expert is critical. The qualities of a good expert are communication ability, honesty, fluency, and commitment to the project (33:4-7). A key to design, training, and evaluation of the system is the selection of real test cases (33:4-9).

## Software and Knowledge Engineering Research Reviewed

Several researchers have contributed to progress in the area of automated software tools. In particular, the research projects of Abrett and Burstein, Shapiro and McCune, Harandi, and Waters discussed in the following section provided input into the conceptual and detailed requirements of the prototype knowledge editor as shown in Chapters III and IV, respectively.

### Abrett's and Burstein's Work on KREME.

Glenn Abrett and Mark Burstein of BBN Laboratories believe that "one of the major bottlenecks in large-scale expert system development is the problem of knowledge acquisition: the construction, maintenance, and testing of large knowledge bases" (1:1). They state that forcing all types of knowledge into a particular representation formalism is difficult and time-consuming (1:1). Resulting from these problems, BBN developed the KREME (Knowledge

Representation Editing and Modeling Environment) system which includes a frame editor, rule editor, and procedural editor. Furthermore, KREME provides macro-editing facilities ensuring maintained consistency and confidence in the knowledge base. These facilities are espcially important as the project increases in scale (1:12). KREME also includes an easy-to-use mouse driven system, providing several data accessing methods (1:5).

In their frame editor, knowledge is represented using slots such as: 1) role restriction, called constraints by Geddes, 2) disjoint, called side-effects by Geddes, and 3) equivalances, which can be thought of as synonyms. BBN's procedural editor is used to represent checklist type steps. An important feature provided by the procedural editor is the ordering of steps, that is, specifying which steps must occur prior to other. Their rule editor includes a graphical display facility, enormously aiding in the knowledge representation task.

BBN used object-oriented programming concepts for representing their knowledge. They created instances of the appropriate knowledge objects (frames, rules, or procedures), attaching knowledge to that object. This approach kept the knowledge organized and well partitioned.

Shapiro's and McCune's Work on IPE.

The prototype Intelligent Program Editor (IPE) developed by Daniel Shapiro and Brian McCune of the Advanced Information and Decision Systems (AI&DS) Company was a

knowledge based tool applied to the analysis and manipulation of computer programs (30). The purpose of their research was to build an intelligent editor for assisting the Air Force in maintenance problems.

The Intelligent Program Editor (IPE) consisted of three primary components: the Extended Program Model, the Programming Context Model, and a library of tools for program manipulation. The Extended Program Model contained the knowledge and accessor functions for manipulating the data, while the Programming Context Model provided the references to the current state of the world, putting the user's intentions in their proper context (30:227). Finally, the library of tools included "a collection of semantic analysis and manipulation tools that provide the programmer with a more powerful vocabulary for manipulating programs, above the level of character by character, or line by line changes" is provided (30:227). IPE was developed on a Symbolics LISP machine, providing the user multiple window menus.

Key features of the Extended Program Model were a well-defined vocabulary for communicating with the database and pre-defined database accessor functions for manipulating the contents of the database. Constraint checking was also performed on the consistency and correctness of the data (30:229). The Programming Context Model provided detection of missed steps, implying some data was represented as checklists. The analysis tools included in the editor were

broken into three areas: advanced program manipulation, semantic analysis, and style analysis (30:228).

The power offered by an editor becomes apparent in situations where the details are overwhelming.

> Becoming overwhelmed occurs in the process of editing programs which are too large to remember explicitly, in the act of understanding code which has rarely been seen before, or in the process of completing partially implemented designs. In the context of program maintenance, a search mechanism helps to alleviate some of the burden on the programmer by supplying an intention-oriented vocabulary for referencing code. (30:230)

While building this prototype two major concerns are: the stepwise building of a knowledge base and the recognition of user intentions. Additionally, AI&DS concentrated on a user-friendly interface for gathering the user's knowledge and a template-oriented editor for representing of the knowledge (30:232).

Harandi's Work on KBPA.

Medhi Harandi, a professor of Computer Science, at the University of Illinois at Urbana-Champaign discussed the research being performed on the KBPA (Knowledge-Based Programming Assistant). KBPA assists programmers in these software lifecycle phases: designing, coding, debugging, and testing.

> It implements various elements of programming expertise as an interactive system equipped with provisions by which the domain specialist could easily and effectively transfer to the system the knowledge it needs for its decision making. (20:233)

Using both artificial intelligence and software engineering techniques, the KBPA expert environment addresses three issues: how to convey the intent of the goal program, how to model the state of the current world, and what knowledge, rules, heuristics, and strategies must be employed. Over the past two decades, a variety of approaches to all three issues has evolved (20:233).

The interactive functions provided by the Assistant include: knowledge extraction, knowledge representation, program design consulting, knowledge-base management, teaching the user, and tutoring the user (20:234). Each of these functions will be briefly discussed.

1) Knowledge extraction is performed when an inconsistent or incomplete data flow model compared to the specification is detected by the system. "Where appropriate, the system requests additional information and/or informs the user of any problems" (20:236).

2) Knowledge in the system is hierarchically represented in a framelike structure, laying the foundation for inheritance concepts (20:235).

3) Interactive consulting is provided in the design, coding, and debugging phases of a task (20:234).

4) Internal management of the knowledge base is provided through general query and update commands (20:234).

5) "By generating explanations and descriptive text from the knowledge base the system can be used as a teaching device" (20:235).

6) By learning from previous errors, the system can continuously expand its knowledge and capabilities (20:235).

Harandi mentioned the difficulty in writing rules, primarily due the difficulty in interviewing an expert to acquire the knowledge. He believes the complexity of the knowledge engineering task arises because experts cannot explain their reasoning process. After performing a particular job for an extended period of time, the process is forgotten (20:237). As a result, Harandi believes it is easier to teach an expert how to use a knowledge editor, than it is to teach a knowledge engineer the expert's domain, if given a graphical, well-designed computer tool. To reach his goal of sitting the expert down with a knowledge editor, Harandi stated two aspects of knowledge acquisition he has concentrated on:

> 1) the capability of composing, entering, updating, modifying, and testing rules in a highly interactive and user friendly environment; and
>
> 2) the ability to generate rules from examples. (20:237)

Overall, review of the initial work on the KBPA editor for the purpose of acquiring knowledge, when supported by graphical displays, has been positive (20:238).

Waters' Work on KBEmacs.

Richard Waters, also working on an intelligent programming aid, calls his editor KBEmacs (Knowledge Based Emacs). KBEmacs is an initial step towared building an Ada programming assistant called the Programmer's Apprentice. Waters used artificial intelligence concepts in building the KBEmacs editor (35:47).

Waters ensured the editor commands were not overly complex and detailed, redundant, or semantically awkward (35:48). His goal in building the editor was to increase the reliability of the resulting computer program or knowledge base (35:48).

Waters expressed three beliefs about programming. First, due to human limitations, tools for developing large systems must be provided. Second, creativity must be limited if higher productivity of computer programmers is the goal. Third, "when constructing a program, it is usually better to construct a reasonable program rapidly than to construct a perfect program slowly" (35:50).

Waters also believes that the keys to using AI in any domain are: the establishment of a set vocabulary, the selection of proper knowledge representation structures, and the establishment of data accessor functions (35:50).

In his conclusion, Waters stated that "experimentation revealed that programmers were much happier thinking in terms of program text and were somewhat confused by plans" (35:56). Therefore, Waters modified his user interface,

abstracting away the plans inherent in the program. The importance of the user interface was demonstrated in this application.

Software and Knowledge Engineering Summary

The authors of the works reviewed provided an automated tool for accomplishing a task; they agree on the importance of the user interface in acquiring knowledge, and they agree that properly representing the knowledge is difficult, yet important for a flexible, expanding system. Furthermore, pre-defined knowledge accessor functions must exist, as well as an agreed upon vocabulary for the specific domain. Both software engineering and knowledge engineering principles must be adhered to throughout the task. The author used the experiences of the researchers in Chapter II to develop the conceptual design of the F-16 knowledge editor, which will now be presented in Chapter III, "Conceptual Design."

# III. Conceptual Design

## Introduction

A conceptual design includes the high-level 'how' as distinct from the internal structure and processing details. In his book on software engineering, Richard Fairley calls this high level of design abstraction the external design (13:137).

> During software design, abstraction allows us to organize and channel our thought processes by postponing structural considerations and detailed algorithmic considerations until the functional characteristics, data streams, and data stores have been established. (13:139)

This section contains a high level, abstract, structural overview of the rule editor, postponing the detailed considerations until Chapter IV. The overall goal will be a facility that allows the pilot to specify his understanding of a checklist or flight manual to a computer easily. He accomplishes this goal by specifying actions necessary to implement a script in a knowledge editor.

## Test Case to Derive Editor Features

Using a sample script from an F-16C fighter aircraft flight manual, for the air-to-ground weapon delivery mode, the necessary knowledge acquisition and representation editor features are derived. The F-16C has eight basic modes for air-to-ground weapon delivery (see Figure 6). The visual modes include strafe (STRF), continuously computed impact point (CCIP), dive toss (DTOS), and visual

| MODE/SUBMODE | | VISUAL | | | | MANUAL | PREPLANNED | | |
|---|---|---|---|---|---|---|---|---|---|
| CONTROLS | | STRF | CCIP | DTOS | EO-VIS | | CCRP | LADD | EO-PRE |
| MASTER ARM | | MASTER ARM/SIMULATE | | | | | | | |
| MASTER MODE | | A-G | | | | | | | |
| CANNED FCR MODE | | AGR | | | | | GM | | |
| AIMING | SELECT STPT | O | | | | | X | | |
| | SELECT TGT/IP/ RP/OAP AIMING | NA | | | | | | | |
| | SLEW CURSOR OVER TGT/AIMPT | NA | | O (HUD) | | NA | | | |
| | FLY TGT SYMBOL To TGT/AIMPT | X | | | | | | | |
| | FLY FPM TO STEERING LINE | NA | | | | | X | | |
| ATTACK | WEAPON RELEASE | | | | | | | | |
| | TRIGGER | X | NA | | | | | | |

X Required switch action
O Optional switch action
NA Not applicable

Figure 6. Air-to-Ground Weapon Delivery Switchology Requirements (19:9-3)

electro-optical (EO-VIS). The pre-planned modes are continuously computed release point (CCRP), low altitude drogue delivery (LADD), and pre-planned electro-optical (EO-PRE). Manual (MAN), the last mode, is neither classified as visual or pre-planned. The mode chosen for the first test case is the CCIP (Continuously Computed Impact Point). The symbology used on the head-up-display (HUD) for the air-to-ground weapon delivery mode is shown in Figure 7.

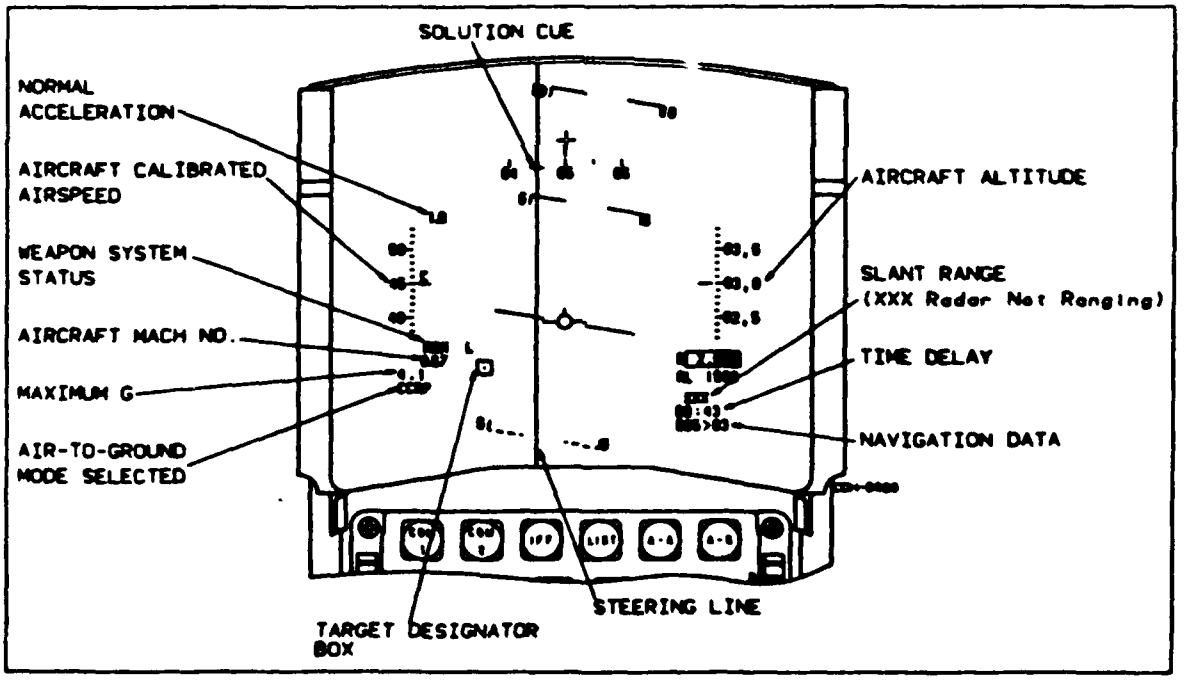



Figure 7. Air-to-Ground Symbology (19:4-106)

CCIP (Continuously Computed Impact Point).

The CCIP checklist includes the following steps (see Figure 8):

1. Access A-G mastermode.
2. SMS - Select weapon and CCIP.
3. HUD - Verify CCIP bombing symbology displayed.
4. MASTER ARM switch- MASTER ARM or SIMULATE (as applicable).
5. CCIP pipper - On target.
6. WPN REL button - Depress and hold.

After Release

1. WPN REL button - Release.
2. MASTER ARM switch - OFF.
3. CCIP mode - Exited (as desired). (19:9-26,9-27;12:23)

Pilot and Flight Manual Details of Script.

The details of this script and when it is applicable have been taught to the F-16 pilot in his specific training. This training was not given to another pilot or to a non-flyer; therefore, their level of understanding of the script is not the same. Furthermore, each F-16 pilot's understanding of the script, including the type of information he would like to see as a part of it differs. Every pilot is an individual with a particular set of experiences tucked away in his personal knowledge base, and as an individual should be able to tailor the aircraft systems to best aid his needs (9). Furthermore, explanation of the thought

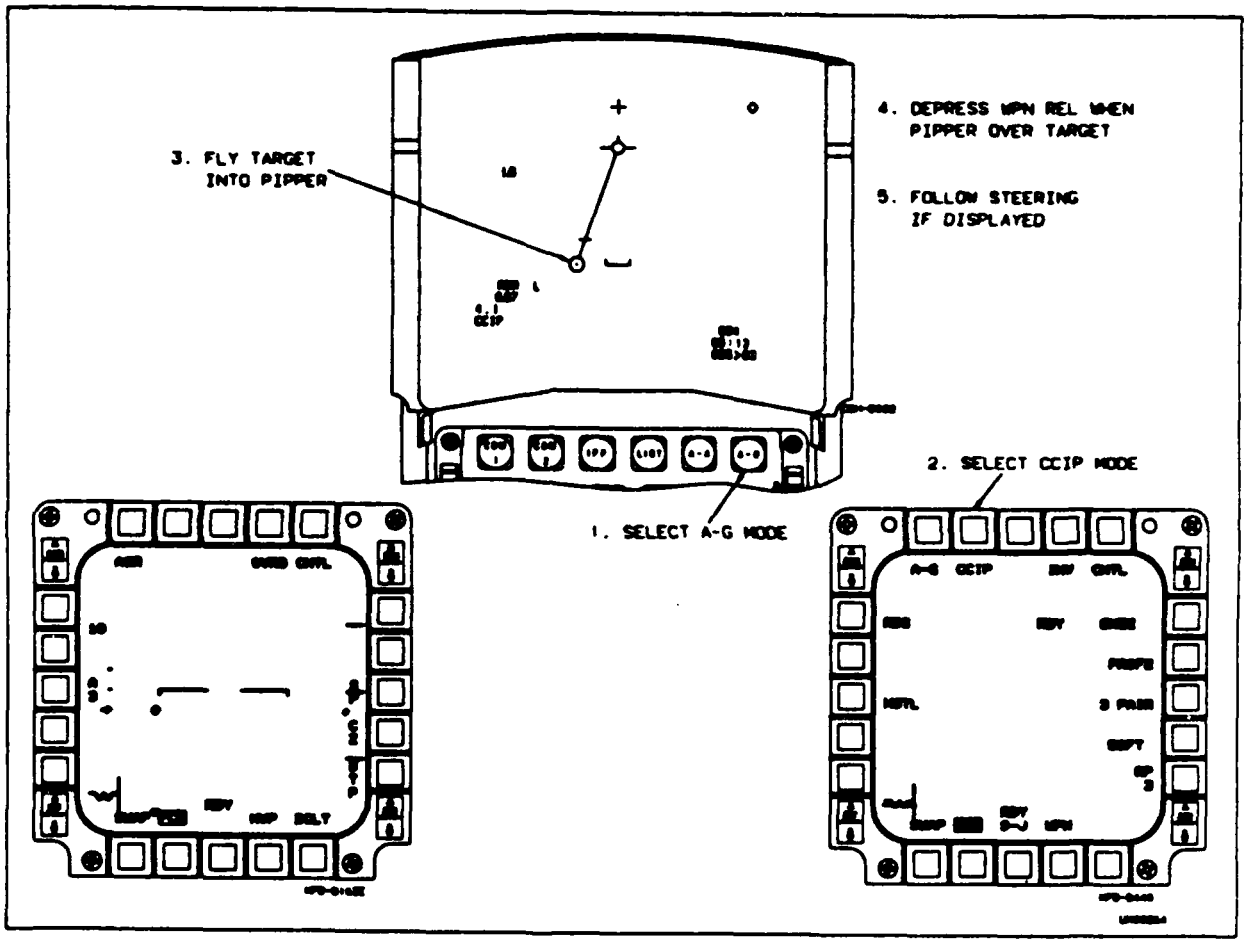



Figure 8. CCIP Bombs Operation (19:9-26)

process used by an expert is difficult to verbalize, especially when the requester (knowledge engineer) is not trained in the expert's specific field. As a result, the knowledge entered into an expert system, initially, should be done jointly by the pilot and the knowledge engineer. Only the knowledge engineer knows the intricacies of knowledge representation and only the pilot knows his domain, preferences, and experiences. A tool which hides all the inferencing and control strategies, and details of the knowledge representation, however, should not require a knowledge engineer, once the pilot has built up confidence in the tool.

Interviewing the pilot to extract the details hidden between the lines in a checklist script symbolizes the pilot's understanding of when and where to apply one script versus another. His preferences on desired constraints or the on the order of checklist steps can be expressed at this time, thus tailoring the script. The pilot obtains these details from experience and training.

The following text shows some of the CCIP checklist details, as given in the F-16 flight manual and/or by Major Dick Frank, the F-16 pilot used as the domain expert for this research. The CCIP mode computes and displays weapon impact point on the HUD (head-up-display). The pilot accesses the CCIP mode by depressing the A-G (air-to-ground) button on the ICP (integrated control panel), selecting the desired weapon profile, and finally selecting CCIP mode on

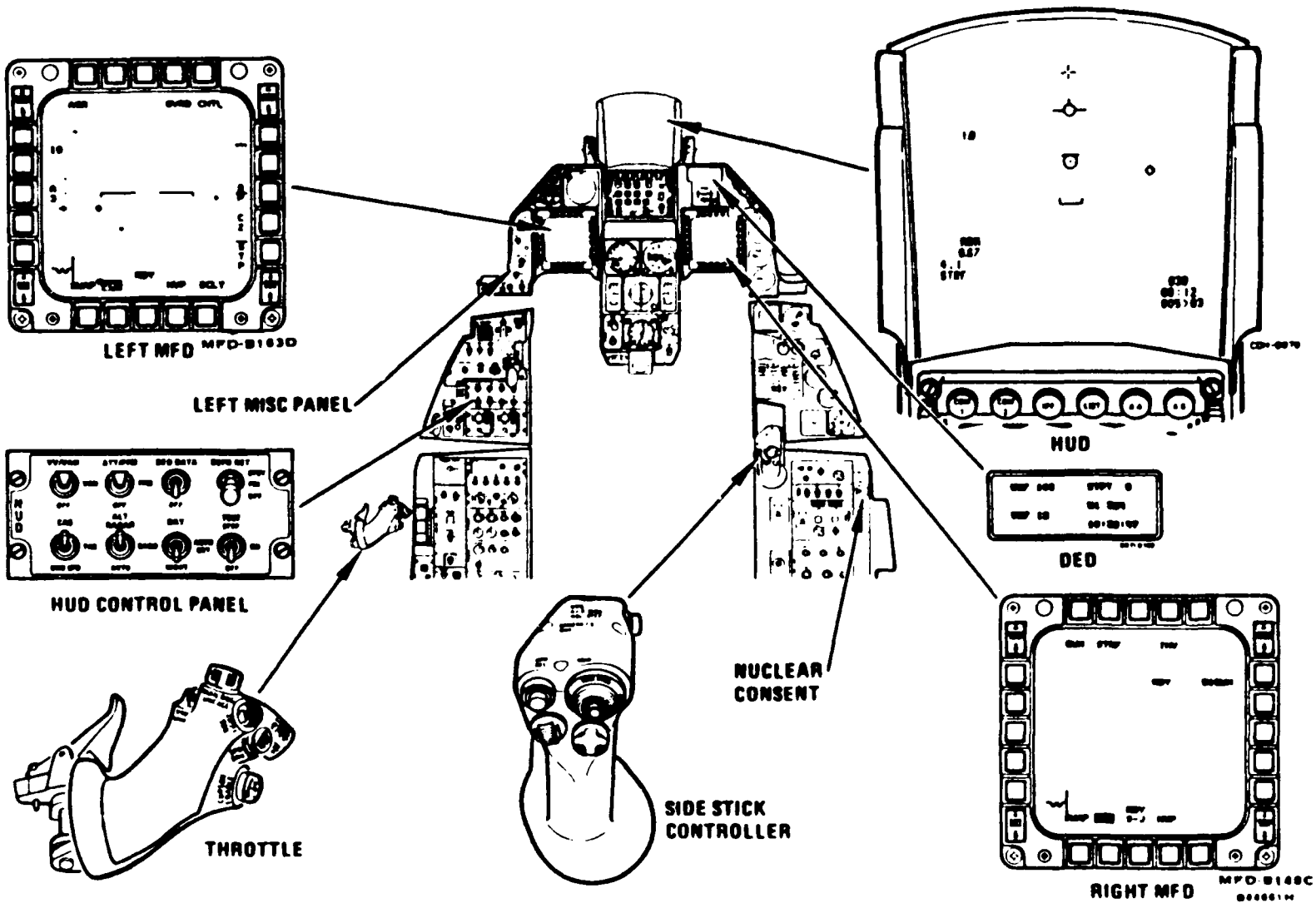



Figure 9. Air-to-Ground Attack Controls and Displays (19:92)

the MFD (multi-function display). The layout controls and displays in the F-16C cockpit are shown in Figure 9. When the pilot selects this mode, the aircraft computer provides computation using automatic ballistics to compute the weapon impact point and automatic delayed release, if required. The aircraft also provides computation of the pullup break-away cues for safe ground clearance on the HUD.

The pilot verifies entry into the CCIP mode by recognizing the proper symbology on the HUD display and the letters CCIP on the lower left section of the HUD. In the CCIP mode, the radar defaults to the AGR (air-to-ground ranging) mode versus the baro ranging provided by the GM (ground-map) mode. "AGR provides automatic ranging data in the 10-mile scale for height-above-target measurement" (19:9-6). AGR uses the radar to determine slant range. Placing the MASTER ARM switch in the SIMULATE position results in SIM being displayed on the HUD, and no actual weapon release occurs upon depression of the WPN REL button. However, when the switch is put in the MASTER ARM position, ARM will be displayed on the HUD, and weapons will be released upon WPN REL depression. A visual cue to the pilot signifying that weapons were properly released is a flashing FPM (flight path marker) on the HUD (15). The flashing will continue until the WPN REL button is released. However, the pilot must hold the WPN REL button depressed when he desires ripple delivery. This button should be depressed when the pipper and target symbology on the HUD are superimposed.

During the CCIP weapon delivery mode, some important cues to the pilot are displayed on the HUD. If the aircraft vertical velocity rate of descent is greater than 50 feet per second, a pullup anticipation cue will appear and move toward the flight path marker as altitude decreases. Upon reaching the flight path marker, a large flashing 'X' (breakaway cue) will be displayed on the HUD and MFD's, indicating that an immediate 4g pull-up is required for safe ground clearance (19; 12) (see Figure 10). This maneuver is normally expressed as: when breakaway cue perform 4g pullup in two seconds. This really means that the pilot is allotted one second to pull back on the stick, and the aircraft is allotted one second to respond to the action, thus safely clearing the ground. The pilot can discontinue the 4g ascent any time after safe ground clearance is achieved (15).

For bomb delivery (see Figure 11), a delay cue indicates that a time delay is required. If present, depressing the WPN REL button when the pipper and target are superimposed will only designate the target. In addition to the delay cue, a steering line and time-to-go (TTG) readout will appear on the HUD. While the pilot keeps the flight path marker (FPM) on the steering line, the delay cue moves toward the FPM. When the TTG readout reaches zero, the delay cue will be centered on the FPM and the FPM should be centered on the steering line, automatic weapon release will occur.

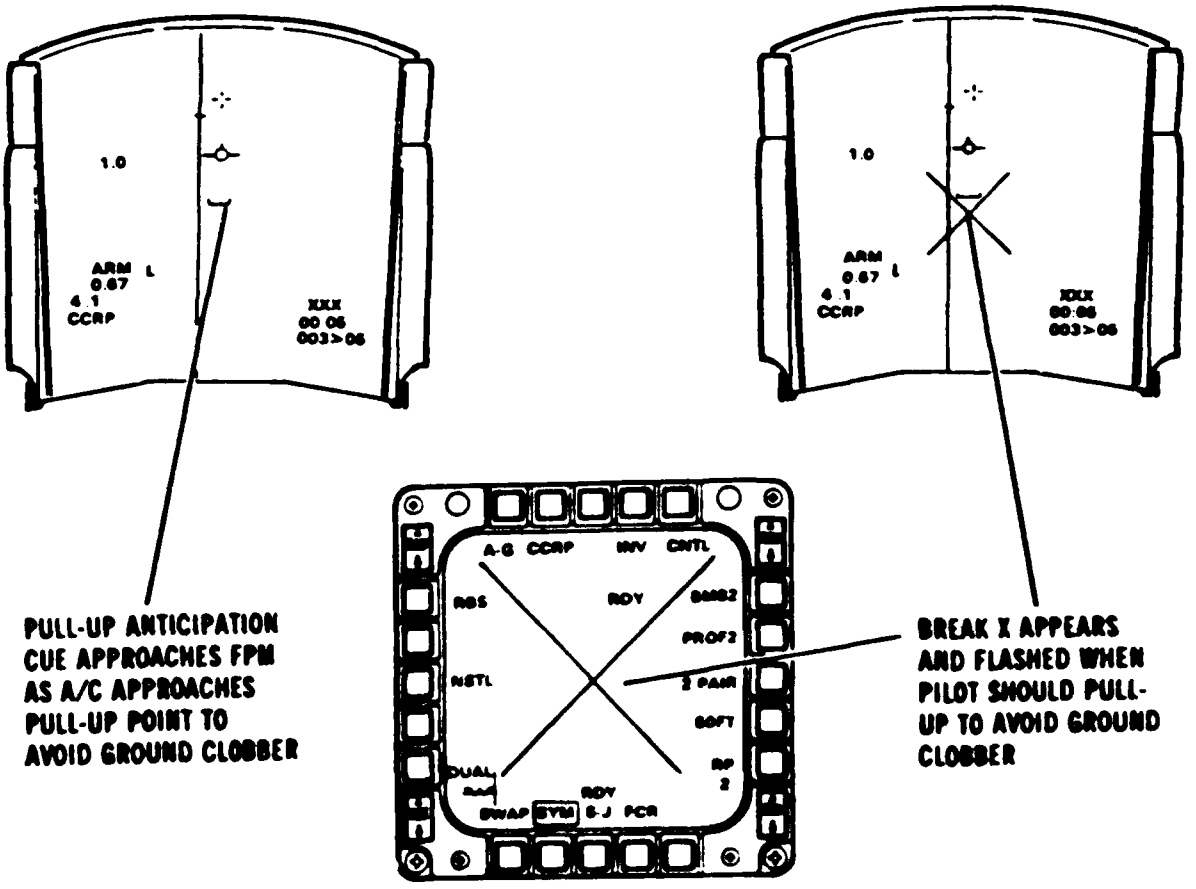



Figure 10. Ground Clearance and Breakaway Cues (12:1-162)

For rocket delivery (see Figure 12), an inrange cue (a small horizontal line) is provided at the top of the CCIP pipper (small circle) when range to impact point is within 8000 feet. A bombfall line is not displayed since rocket launches are over the nose of the aircraft (15). The WPN REL button should not be depressed until the inrange cue appears when firing rockets.

Prior to exiting the CCIP mode, the WPN REL button should be released and the MASTER ARM switch should be set into the OFF position (12).

Rule-Based Representation of Script.

Before the author converts the script into an unambiguous frame format usable by a computer, she will demonstrate a rule-based intermediate format. A rule consists of 'if' and 'then' parts, so that IF a condition holds true, THEN its action occurs. Sometimes one or more obstacles, called "constraints," may exist on the condition. A constraint to the delivery of weapons may be that the angle of attack of the aircraft is too high or that the remaining fuel is not sufficient to perform the maneuver. Besides constraints on the IF part of the rule, there are side-effects on the THEN part. A "side-effect" is the unspecified result of executing the THEN part of a rule, after properly matching the IF part. For example, if the pilot switches from the CCIP mode to the CCRP mode, the side-effect of activating the CCRP mode is the deactivation of the CCIP mode since it is not possible to be in both

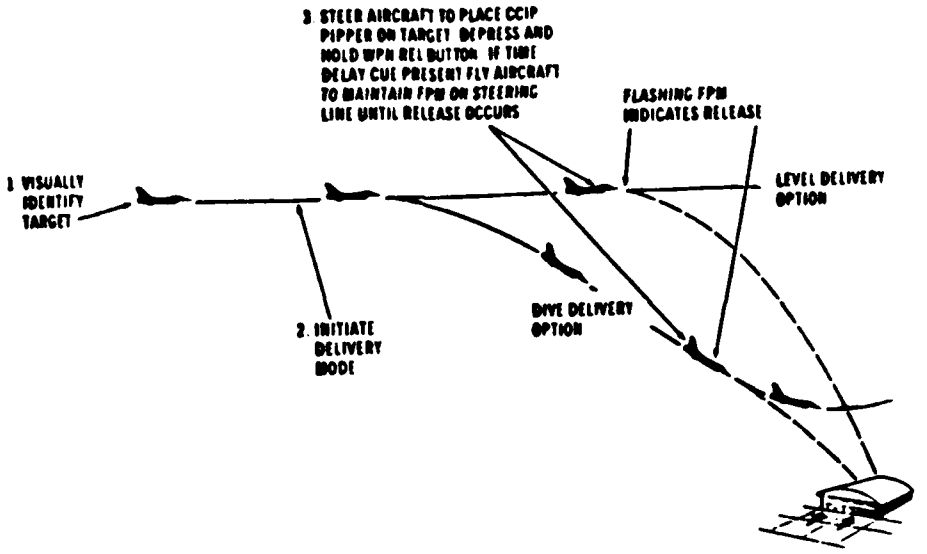



Figure 11. CCIP Bomb Delivery Maneuver (12)

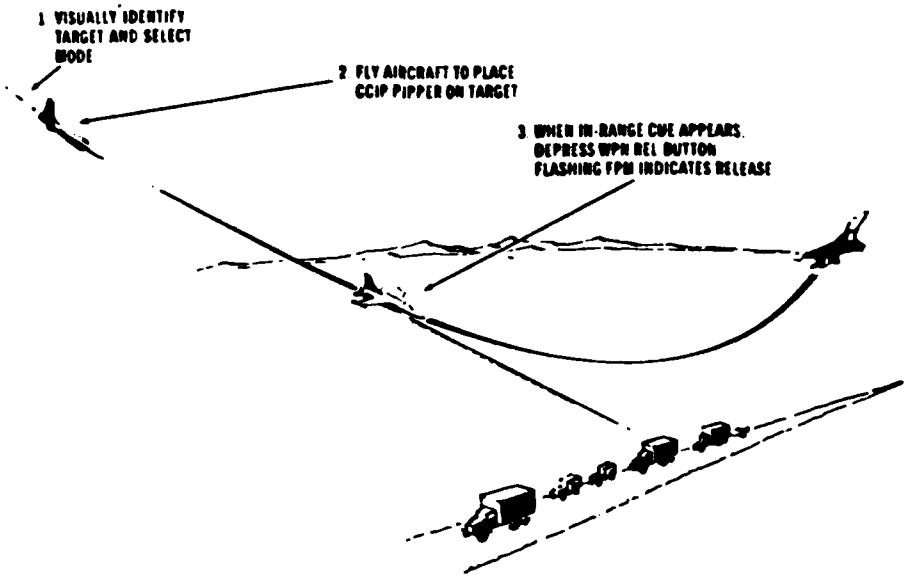



Figure 12. CCIP Rocket Delivery Maneuver (12)

modes simultaneously. "Typical side-effects are activation or closure of scripts, removal of plans or goals, or immediate tests of other rules" (16:166).

Converting the CCIP checklist script into the rule-based format produces:

RULE

IF delivery mode = CCIP script,

* And if these CONSTRAINTS are satisfied

IF vertical velocity descent > 50 feet/second,

THEN pullup anticipation cue present. (19:9-20)

IF pull-up cue,

THEN 4g pull-up in 2 seconds. (One second for the pilot to respond by pulling up on the stick, and one second for the aircraft to respond, safely clearing the ground). (19:9-20; 15)

IF breakaway cue,

THEN execute immediate 4g pull-up.

THEN depress A-G mode button on ICP (ON),

THEN select CCIP mode on MFD (rotary),

* And if this CONSTRAINT is satisfied

IF delivery mode = CCIP and IF weapon(s) selected on SMS (stores management subsystem),

THEN set MASTER ARM switch to ARM or SIM.

* And if this CONSTRAINT is satisfied

IF rockets selected,

THEN don't depress WPN REL until inrange symbol appears (less than 8000 feet).

THEN depress WPN REL button,

* And if these CONSTRAINTS are satisfied

IF time-delay set,

THEN do not release WPN REL until TTG = 0.

IF ripple release,

THEN do not release WPN REL button until done.

* SIDE-EFFECT of last action

IF MASTER ARM = ARM,

THEN weapons released.

THEN release WPN REL button,

* SIDE-EFFECT of last action

Set MASTER ARM = OFF.

Exit CCIP mode.

Why an Internal Frame Representation of Script is Needed

A set of conventions to be used in describing any theory or thing is important for establishing a solid building foundation. According to Dr. Patrick Winston, head of the Department of Artificial Intelligence at the Massachusetts Institute of Technology (MIT):

> Experience has shown that designing a good representation is often the key to turning hard problems into simple ones, and it is therefore reasonable to work hard on establishing what symbols a representation is to use and how those symbols are to be arranged to produce descriptions of particular things. (40:179)

A consistent, unambiguous internal representation of the detailed information in the CCIP script is required if computer understanding, as defined by Schank in Chapter II,

is to be attempted. Transforming the pilot's rule-based understanding of a script into a frame representation usable by a computer is perfomed through a knowledge acquisition and representation editor. The editor provides a mouse-driven interface and pop-up menus to the pilot allowing him to enter information about each step in a checklist. Yet, the internal complexity of the editor is hidden from the pilot.

The data structure used for representing the stereotypical F-16C checklist scripts is called a frame. A frame can be viewed as a network of nodes and relations (40:180). A frame consists of many slots that are filled (instantiated) with data particular to the script being represented. For example, in describing an airplane, the frame could contain slots for number of engines, type of engine, gross weight, maximum speed, and typical weapons on-board. The frame and slots representing this description are:

```
((airplane  X-33)
   (number-of-engines  2)
   (type-of-engine  F-100)
   (gross-weight  20000)
   (maximum-mach-speed  3)
   (typical-weapon  sidewinder))
```

These slots make up the pieces of each rule in a script. Chapter II pointed out that the slot filling process was the basis of Minsky's frame theory.

The types of slots anticipated in the rule editor follow

the format used by Schank and Abelson. These include slots for: ACTION, ACTOR, OBJECT, and DESTINATION (both TO and FROM) (29). Additional slots were determined necessary during the prototype implementation. These slots will be discussed in Chapter IV, "Detailed Design."

In the F-16 domain, a slot for specifying which dial, switch, button, throttle, or stick will be accessed was required. Appropriate for such a function is the OBJECT slot, since it is modelling the object to be manipulated by the ACTOR. The editor convention for specifying the desired object and actor is illustrated as:

```
(Object (Panel MFD) (Button CCIP))
(Actor (pilot))
```

The first statement specifies that the object of interest is located on one of the multi-function displays, and in particular, this object is the option select button labelled CCIP. The second statement shows that the person or thing performing the action is the pilot.

For the user, to tell the computer that a switch is moved from one position to another, the slot location is provided.

```
(Object (Panel LEFT-MISC) (switch MASTER-ARM))
(Location (From ARM) (To OFF))
```

These two statements show that the master arm switch, located on the left miscellaneous panel, is moved from the ARM position to the OFF position. Updates to these values

are handled by the editor, without the user needing to specify an action such as : (Erase MASTER-ARM = ARM).

Since different objects require different actions performed upon them, a set of action verbs was defined for each object. For instance, the ACTION that can be performed on a button by an ACTOR is to depress it or release it, whereas a switch requires the set action. Additionally, valid positions for these objects are also be pre-defined. These actions and locations are specified in the detailed design, Chapter IV.

Why a Knowledge Editor is Needed

"One of the most time consuming tasks in building knowledge bases is maintaining internal consistency" (1:9). To maintain internal consistency for a script, the pre-defined panels, objects, and settings must be available for user access. Furthermore, each time a script is saved, the main knowledge base must be updated for future use. Modifying the contents of slots can impact the entire frame-work of the system, particularly when the knowledge base becomes very large. "Consequently, the size and complexity of knowledge bases is [sic] limited by the extent to which automatic means are provided for consistency checking" (1:9). Thus, the prototype knowledge editor built in this thesis effort was also responsible for maintaining the consistency in the knowledge base, no matter how large it becomes.

Knowledge Editor Overview

Automated tools should provide assistance to the user, while hiding the internal details, for example, through a user's HELP menu. Through the editor interface to the pilot, the actions necessary to implement a script in this checklist are specified. At the highest level, this interface must include features for creating new scripts, editing old scripts, saving a new or updated script, viewing a script, and listing the user-defined scripts.

Conceptually, there are three major modules representing the knowledge editor features. These modules include a user interface with HELP menus assisting in the derivation of script information, the generation of an internal frame representation for a script, and a LISP language code generation of the script (see Figure 13). Output representations from the internal frame representation of a pilot's script, other than LISP code,

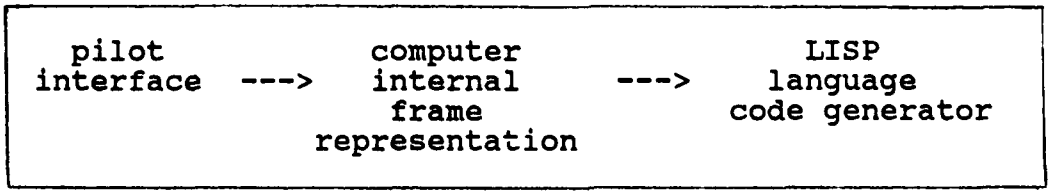



Figure 13. Editor Overview

could be generated. The LISP code generation was implemented demonstrating the potential value of a solidly designed frame representation. Expanding the computer internal frame representation into other possible forms

includes: an English sentence format generator, an Ada language code generator, Geddes' representation code generator, or anyone else's desired representation format (see Figure 14).

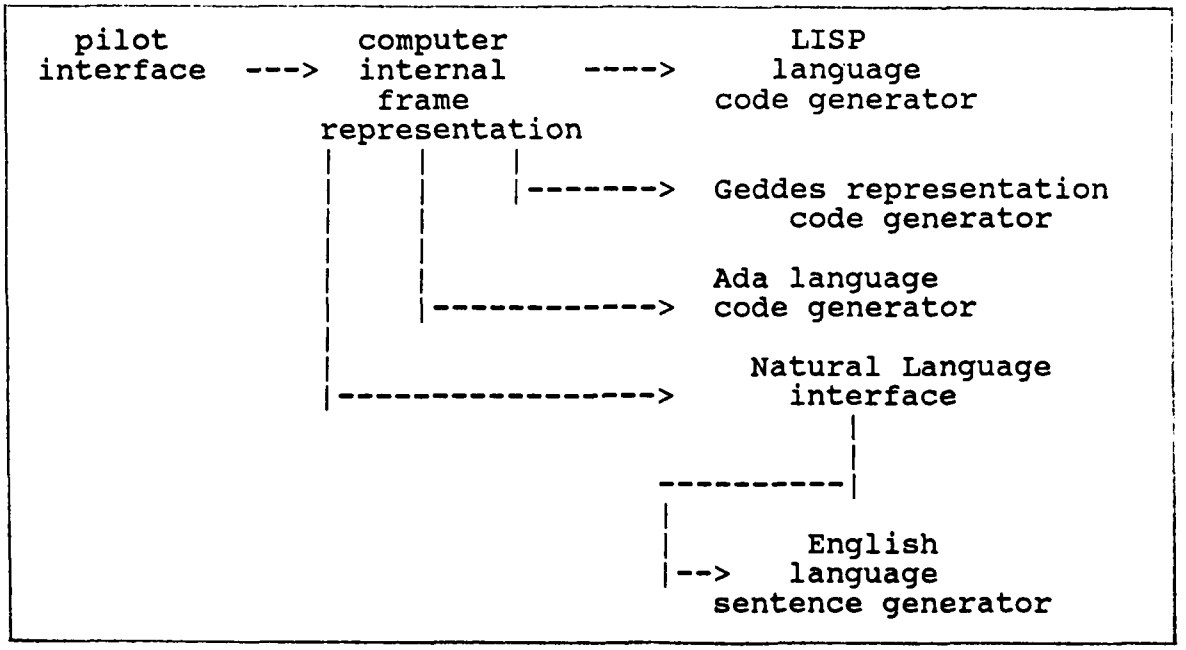



**Figure 14. Possible Editor Expansion Features**

There are four potential benefits of an automated knowledge acquisition and representation editor. First, the pilot will be 'writing' computer programs without knowing the target computer language. Second, the knowledge engineering dialogue with the pilot can be replaced by direct pilot usage of the tool, if the complexity of the internal frame representation of a script is kept hidden from the pilot. Third, if a different implementation

language or format is required, the pilot's knowledge need not be re-entered. All that is needed is an additional output language code generator. Fourth, the executable computer programs produced could be transferred onto a plug-in cartridge into an aircraft, thus tailoring it for each pilot (9).

Summary of Conceptual Editor Requirements

Accomplishing the above evolutionary steps, thus bridging the pilot's understanding of a script into a representation usable by a computer, requires an editor tailored for a pilot so he can enter his checklist or flight manual information. The editor interface must hide the computer details from the pilot. As previously stated, the overall goal of this thesis was to prototype a knowledge acquisition and representation tool allowing the pilot to specify to a computer the actions necessary to help implement a checklist script. Chapter IV will provide the detailed design of such an editor.

Two Additional Test Cases

Two additional examples of F-16C scripts used in the air-to-ground weapon delivery mode are included to assist in deriving the specifics for the detailed design of the knowledge editor. The scripts chosen are the CCRP (Continuously Computed Release Point) and the LADD (Low Altitude Drogue Delivery).

CCRP (Continuously Computed Release Point).

The CCRP checklist is shown below. Figures 15 and 16 show the corresponding HUD and MFD displays.

1. Access A-G mastermode.
2. SMS - Select weapon and CCRP.
3. FCR format - Select GM, TGP, and type aiming (TGT, OA1, OA2, or IP).
4. HUD - Verify CCRP bombing symbology displayed.
5. ICP/DED - Select steerpoint number.
6. MASTER ARM switch- MASTER ARM or SIMULATE (as applicable).
7. Bombing geometry - Updated. ("means to slew aiming symbology over the target or offset point") (15).
8. HUD FPM - Centered on steering line.
9. WPN REL button - Depress and hold for computed release.

After Release

1. WPN REL button - Release.
2. MASTER ARM switch - OFF.
3. CCRP mode - Exited (as desired). (12:2-64)

The pilot's understanding of this script may look something like this:

The continuously computed release point submode combines a basic coordinate bombing mechanization with sensor sighting functions (radar ground map or target pod) to provide attack cues for pre-planned targets (12:2-64). The CCRP mode uses radar GM (ground map) mode, unlike the

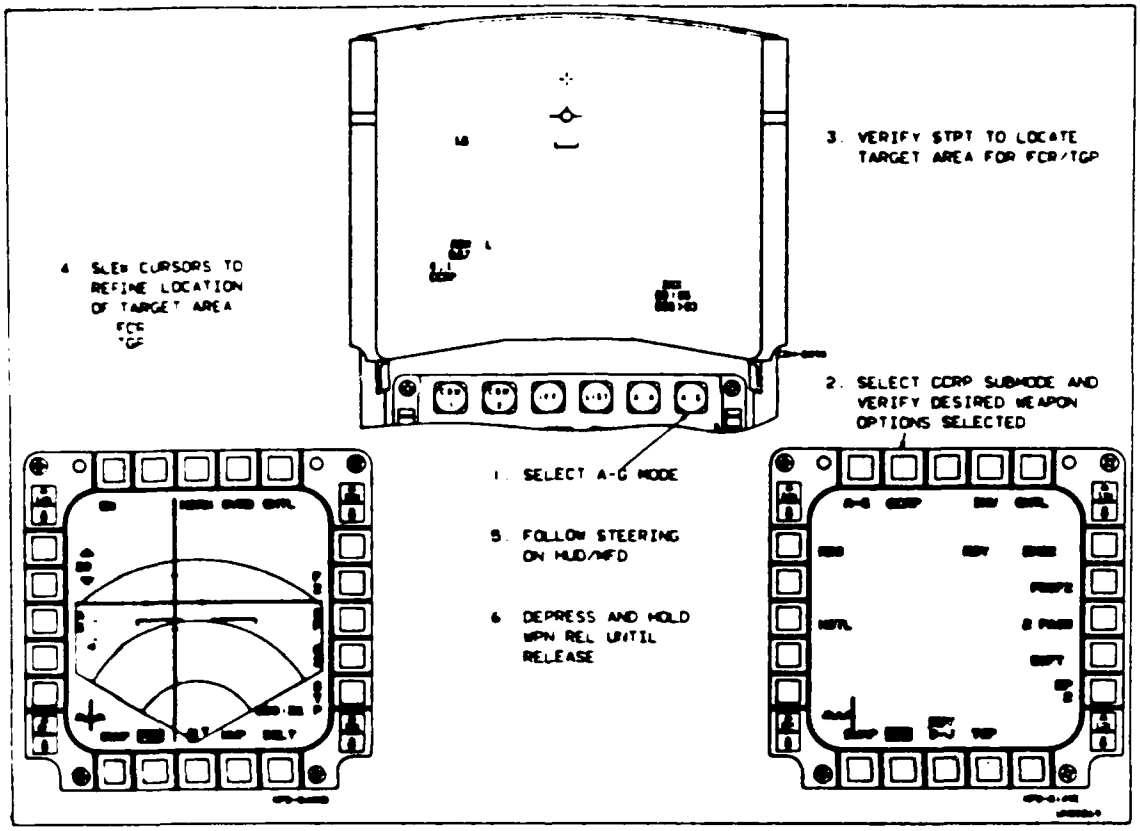



Figure 15. CCRP Operation (19:9-41)

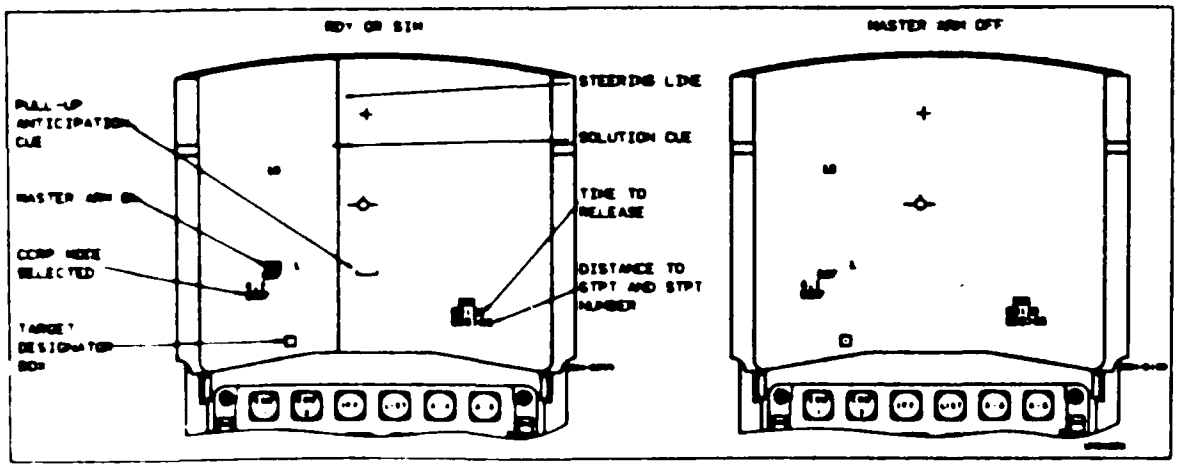



Figure 16. CCRP HUD (19:9-41)

CCIP mode which uses air-to-ground ranging.

According to the F16C-34-1-1 Air Force Manual, "to utilize the CCRP mode, the sensor is slewed to the aimpoint, the weapon release switch is depressed, and HUD steering cues followed to the release point" (19:9-39). The cursor slew is located on the throttle (15). There are 30 possible steering points, all of which may be used as targets. However, offset aiming (OA) may only be utilized with the first 25 of these 30 targets. The desired target number is entered on the upfront controls (UFC), and the target and offset aiming is set up through the FCR (fire control radar) or TGP (targeting pod) format located on the multifunction display (12:1-171,1-172). The UFC consists of both the ICP and the DED (15). An example procedure for data entry on the DED is shown in Figure 17.

The symbology on the HUD consists of the flight path marker, the steering line, and the target position box. The FPM should be lined up with the azimuth steering line as a guide prior to weapon release. "When the target is at maximum toss range, a solution cue is displayed on the HUD, and a 4g pull-up at this time results in a maximum range toss" (19:9-40). The weapon will be released automatically as the aircraft approaches a pitch angle of 45 degrees when the solution cue is centered on the FPM and TTG (time-to-go) equals zero. Since the CCRP mode is a level delivery (see Figure 18), the constraints on vertical velocity descent and pull-up are not applicable as in CCIP (6).

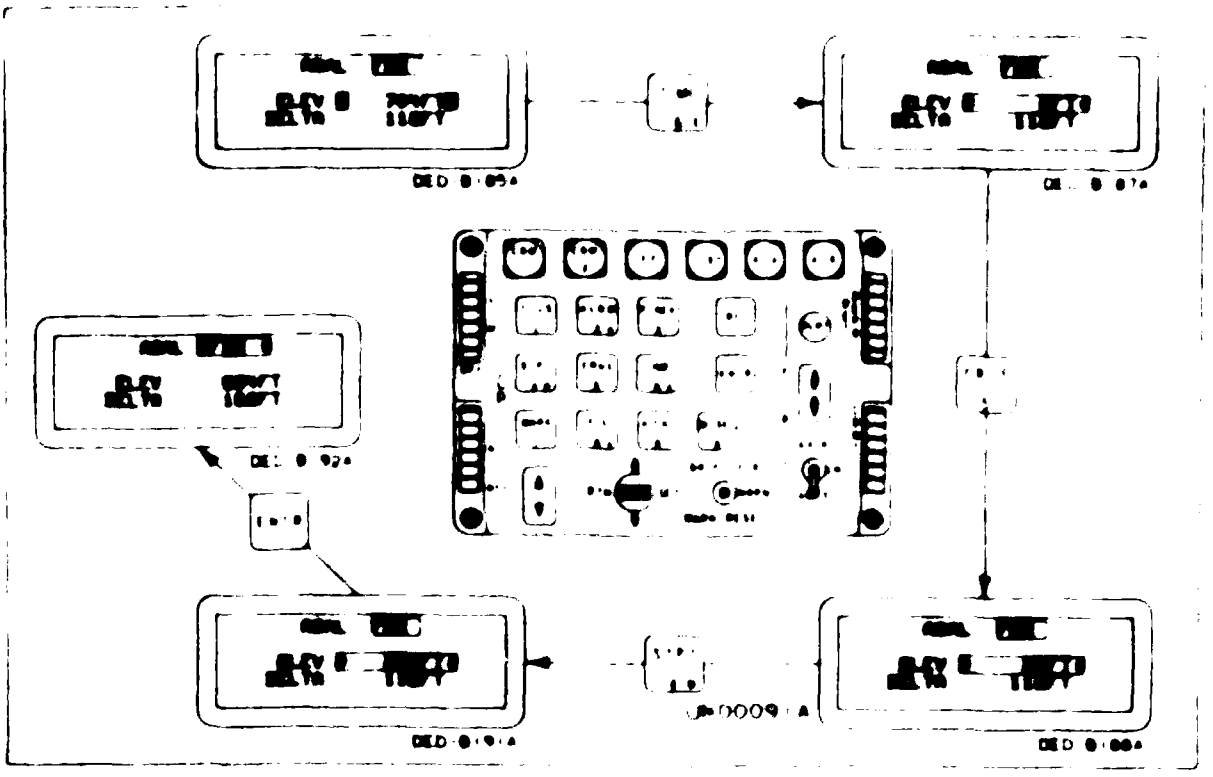


Figure 17. DED Data Entry Procedure (19:4-12)

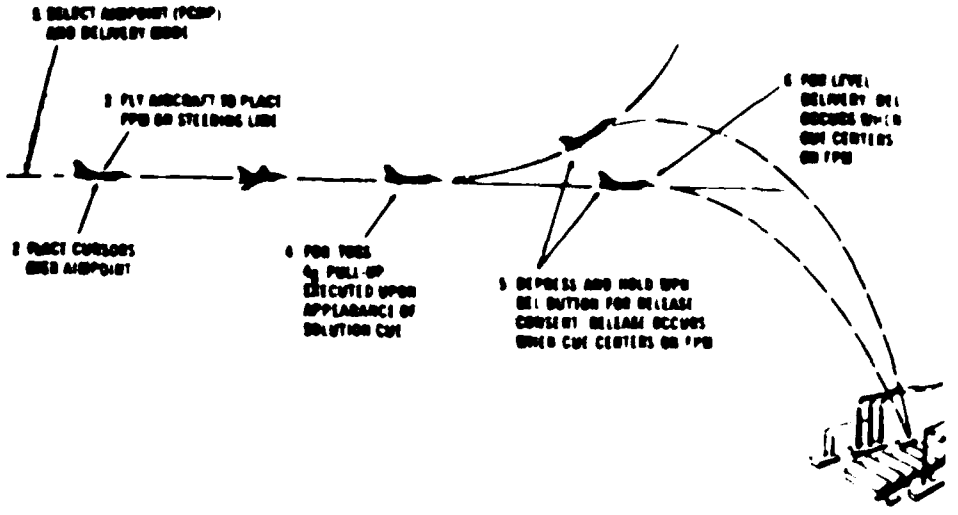


Figure 18. CCRP Maneuver (12)

Now this information is presented in the if-then rule-based format, identifying the constraints and the side-effects of performing the script.

RULE

IF delivery mode = CCRP script,

* And if this CONSTRAINT is satisfied

IF number of g's > 4,

THEN weapon release not cleared.

THEN depress A-G mode button on ICP (ON),

THEN select CCRP mode on MFD (rotary)

THEN select GM (ground map)

THEN select TGP (targeting pod)

THEN select aiming = TGT (target) or
OA1 (offset aimpoint 1) or
OA2 (offset aimpoint 2) or
IP (initial point)

THEN select steerpoint # on DED (data entry display)

* And if this CONSTRAINT is satisfied

IF delivery mode = CCRP and IF weapon(s) selected on SMS (stores management subsystem),

THEN set MASTER ARM switch to ARM or SIM.

THEN depress WPN REL button,

* And if these CONSTRAINTS are satisfied

IF time-delay set,

THEN do not release WPN REL until TTG = 0.

IF ripple release,

THEN do not release WPN REL button until done.

* SIDE-EFFECT of last action

IF MASTER ARM = ARM,

THEN weapons released.

THEN release WPN REL button,

* SIDE-EFFECT of last action

Set MASTER ARM = OFF.

Exit CCRP mode.

LADD (Low Altitude Drogue Delivery).

The checklist for the LADD delivery mode is shown below. Figures 19, 20, and 21 show the HUD, MFD's, and maneuver during the LADD weapon delivery.

1. Access A-G mastermode.
2. SMS - Select weapon and LADD.
3. FCR format - Select GM, TGP, and type aiming (TGT, OA1, OA2, or IP).
4. HUD - Verify LADD bombing symbology displayed.
5. ICP/DED - Select steerpoint number.
6. MASTER ARM switch- MASTER ARM or SIMULATE (as applicable).
7. Bombing geometry - Updated.
8. HUD - Fly LADD maneuver.
9. WPN REL button - Depress and hold for computed release.

After Release

1. WPN REL button - Release.
2. MASTER ARM switch - OFF.
3. LADD submode - Exited (as desired). (12:2-64)

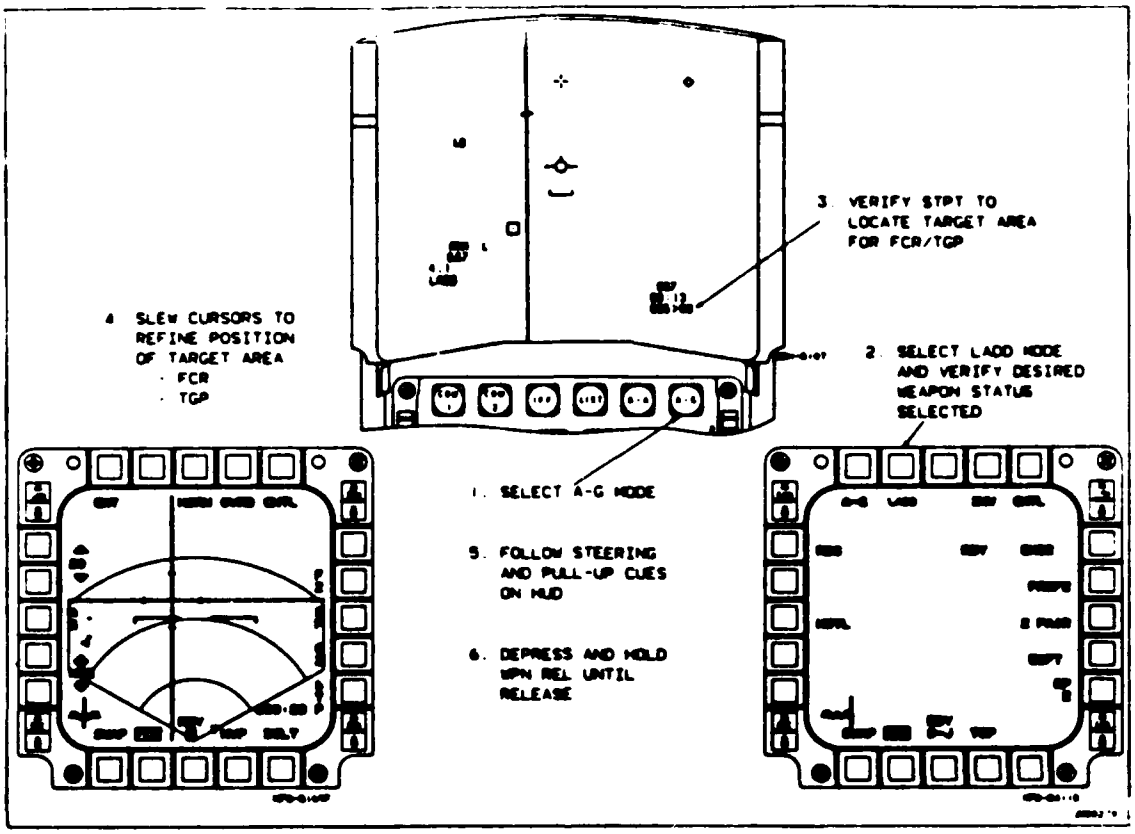



Figure 19. LADD Operation (19:9-45)

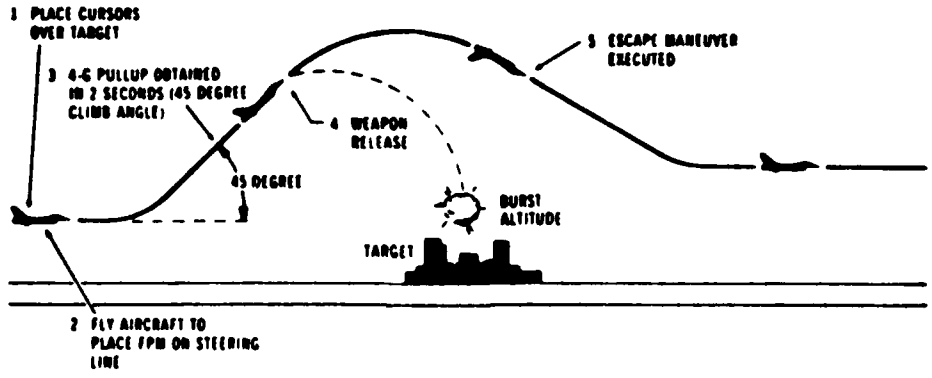



Figure 20. LADD Maneuver (19:9-45)

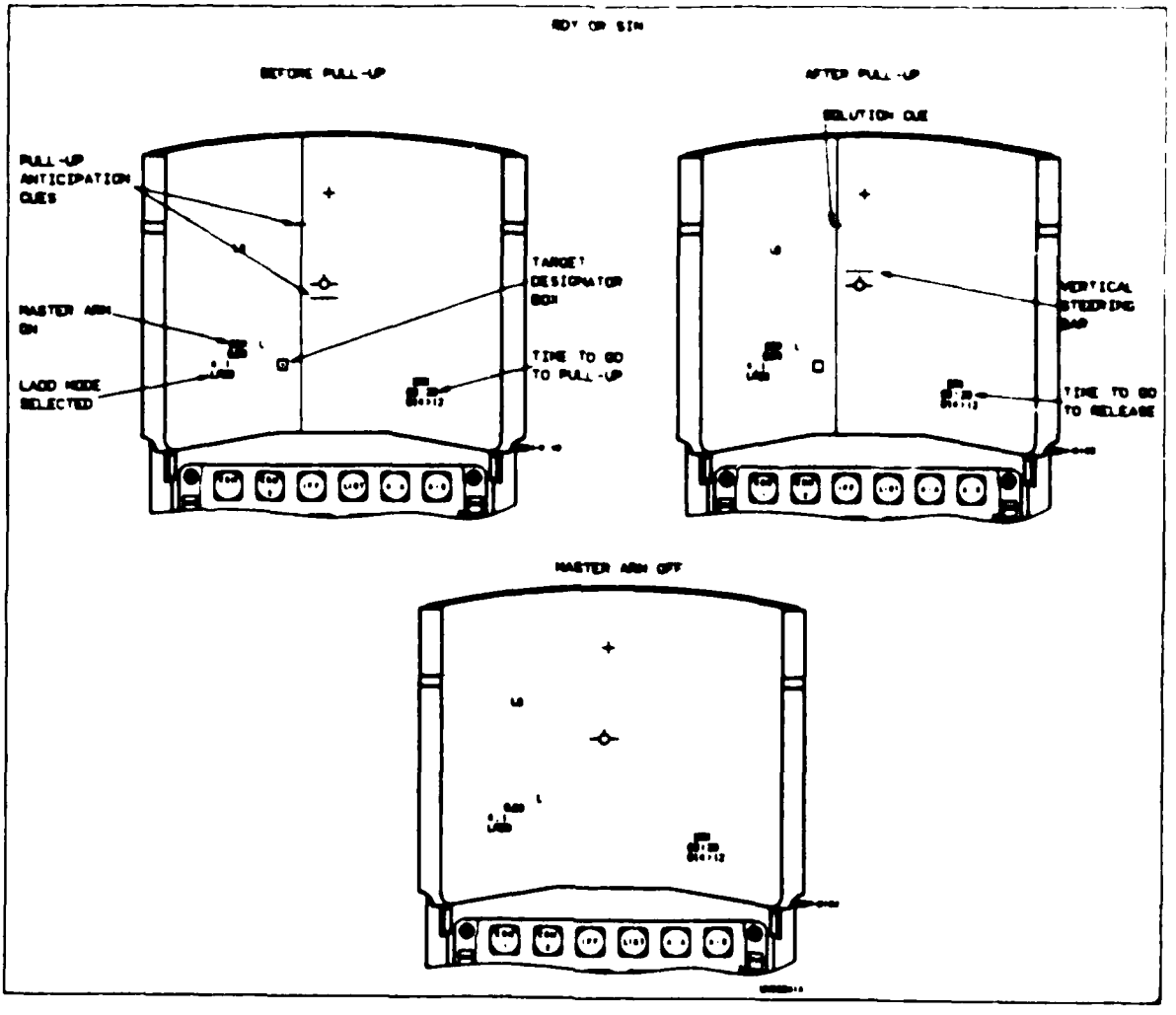



Figure 21. LADD HUD (19:9-46)

The pilot's understanding of the LADD script may be set forth as:

The low altitude drogue delivery submode consists of the same computations, checklist procedures, and displays as the previously discussed CCRP submode. It also provides cues appropriate for the LADD maneuver. This mode is selected when delivering airburst weapons. "It consists of a pull-up to a 45-degree climb, during which the weapon is released, and ther an egress maneuver" (12:1-174). The desired pull-up range is entered via the SMS (stores management subsystem) format control page based on the selected weapon and desired maneuver.

A solution cue is provided, as in CCRI, and originally appears on the steering line above the flight path marker. Pull-up time occurs when the two are superimposed. Unlike the CCRP mode, a maximum toss cue does not appear for the LADD maneuver. After pull-up, if the pilot has depressed and held the WPN REL button on the sidestick, the weapon is released automatically (19:1-174).

When this information is broken into the if-then rule-based format, identifying the constraints and side-effects of performing the LADD script, the result appears as follows:

RULE

IF delivery mode = LADD script,

* And if this CONSTRAINT is satisfied

IF number of g's > 4,

THEN weapon release not cleared.

THEN depress A-G mode button on ICP (ON),

THEN select LADD mode on MFD (rotary)

THEN select GM (ground map)

THEN select TGP (targeting pod)

THEN select aiming = TGT (target) or
OA1 (offset aimpoint 1) or
OA2 (offset aimpoint 2) or
IP (initial point)

THEN select steerpoint # on DED (data entry display)

* And if this CONSTRAINT is satisfied

IF delivery mode = LADD and IF weapon(s) selected on SMS (stores management subsystem),

THEN set MASTER ARM switch to ARM or SIM.

THEN depress WPN REL button,

* And if these CONSTRAINTS are satisfied

IF time-delay set,

THEN do not release WPN REL until TTG = 0.

IF ripple release,

THEN do not release WPN REL button until done.

* SIDE-EFFECT of last action

IF MASTER ARM = ARM,

THEN weapons released.

THEN release WPN REL button,

* SIDE-EFFECT of last action

Set MASTER ARM = OFF.

Exit LADD mode.

Similarities Between the Air-to-Ground Weapon Modes

For the air-to-ground attack mode, the typical pre-planned weapon releases consist of the following similar steps:

* Select the A-G master mode on the ICP (Integrated Control Panel).
* Select the specific submode via the SMS (Stores Management Subsystem) format on the MFD (Multi-function Display), which can be preprogrammed.
* Refine the steering via the selected aimpoint (or overfly the initial point.
* Follow the attack steering on the HUD.
* Depress and hold the weapon release button. The bomb is released automatically when the FCC (Fire Control Computer) determines it will reach the target. (12:1-156)

Once the A-G master mode is selected, the missile step switch on the sidestick can be used to sequence through the other available modes. For the electro-optical weapons, this rotary action includes only the visual (EO-VIS) or the pre-planned visual (EO-PRE) (12:1-156). Many of the constraints and side-effects are the same for these modes of weapon delivery. The CCRP and the LADD checklists are almost exact duplicates. Even the sighting options and bombing geometry updates are the same. "The only differences are the LADD vertical steering guidance and any special procedures (if any) associated with the desired weapon (nuclear consent switch on, nuclear weapon armed)" (19:174).

# IV. Detailed Design

## Introduction

This chapter expands the conceptual design of the knowledge editor into a detailed design. The CCIP (Continuously Computed Impact Point), CCRP (Continuously Computed Release Point), and LADD (Low Altitude Drogue Delivery) weapons delivery checklists already presented will be used to derive the specific knowledge editor features for the F-16 flight domain.

The three main modules comprising the knowledge editor were the pilot interface, the computer's rule-based frame representation format of the pilot's knowledge, and the LISP language code generator, covered briefly in Chapter III. This chapter includes a discussion on the knowledge editing lifecycle, followed by a detailed presentation on each of the three knowledge editor modules.

## Knowledge Editing Lifecycle

Figure 22 illustrates the knowledge editing life cycle. First the expert enters into a dialogue with the computer to provide the rules he uses in his domain of expertise. After acquisition, these rules are transformed into a knowledge representation format for storage into the knowledge base. The knowledge is applied against an application specfic problem to determine the level of computer accuracy in providing an expert solution. The iterative process continues as long as the expert knows something that the

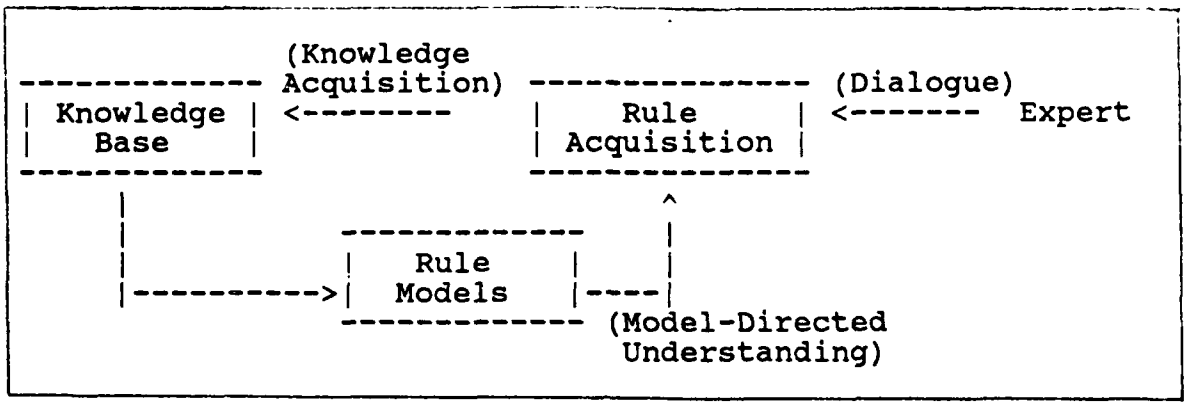



Figure 22. Feedback Loop of Model-directed Understanding and Learning by Experience (5:205)

system does not know. The expert must fill in these gaps by entering additional rules or modifying old rules (5:205).

Pilot Interface

The pilot interface is tied in very closely with the rule-based frame representation format, since the interface is designed to gather the information needed for the internal knowledge representation. To smooth the flow of the pilot's expert knowledge about the flight domain into the computer, a menu-driven interface was developed.

The initial types of slots anticipated in the knowledge editor followed the format used by Schank and Abelson. Included in their system were slots for: ACTION, ACTOR, OBJECT, and LOCATION (both TO and FROM) (29). While attempting to represent the CCIP script, additional slots proved necessary. To sufficiently represent the CCIP script, new slots were created for: PANEL, FEEDBACK, CONSTRAINT, SIDE-EFFECT, SUBUNIT or SCRIPT, and PTR-ORDER.

PTR-ORDER is not displayed in the output files, but is instead used only to correctly order the checklist items (subunits) in each script prior to output. The editor automatically formats the knowledge into slots with the instantiated value and parentheses, such as (Actor (PILOT)) in the frame representation module. Each of the types of slots and their allowable values follow:

1) SUBUNIT or SCRIPT. The subunit slot allows the pilot to name the script or script component (subunit) for referencing purposes. An example of the SUBUNIT and SCRIPT slot is:

(Script (CCIP))
(Subunit (CCIP-1))

CCIP identifies the entire checklist discussed in Chapter III, while CCIP-1 is one of the individual checklist steps.

2) ACTION. The action slot allows the pilot to describe the verbs the computer must know to understand the pilot's actions while flying. Sample verbs include:

* set (for dials and switches)
* depress (for buttons)
* release (for buttons)
* select (for dials, switches, and generic tasks like selecting weapons, CCIP submode, or steerpoint #)
* move (for hands-on controls)

The frame representation of an action is:

(Action (SELECT)).

The editor provides a framework allowing the pilot to define

any additional actions the pilot deems necessary.

3) ACTOR. The actor slot specifies who is responsible for performing a particular action. The only values allowed in the ACTOR slot are:

* pilot
* computer

The frame format is: (Actor (COMPUTER)) or (Actor (PILOT)). The pilot may not define any other actors.

4) OBJECT. The OBJECT slot specifies which button, switch, or other control is accessed by the pilot or computer, as specified in the actor slot, to accomplish an action. Based upon the CIP, CCRP, and IADD weapons delivery modes, the following objects currently exist in the editor's knowledge base for use by the pilot in representing a script:

* A-G Mode Button
* AG Submode OSB (Option Select Button)
* Master Arm Switch
* Missile Step Button
* WPN REL (Weapon Release) Button
* Fire Control Radar (FCR) OSB
* Weapon (Stores) OSB
* Target Pod (TGP) OSB
* Throttle
* Side-Stick
* Trigger
* Steerpoint Rocker Switch

* Radar Mode OSB

* Aiming OSB

The editor provides a framework allowing the pilot to define any additional objects. The object slot incorporates the panel slot as shown:

(Object (Panel MFD) (Button AG-SUBMODE-OSB))

5) PANEL. The panel slot, subsumed by the object slot, can be used in pin-pointing the physical location of an object, such as a switch or a button, thus allowing interrogation of the proper computer subsystem.

* Integrated Control Panel (ICP)

* Multifunction Displays (MFD)

* Data Entry Display (DED)

* Upfront Control (UFC)

* Hands On

* Fire Control Panel

* Left Miscellaneous (Left Misc)

The editor provides the framework allowing the pilot to define any additional panels.

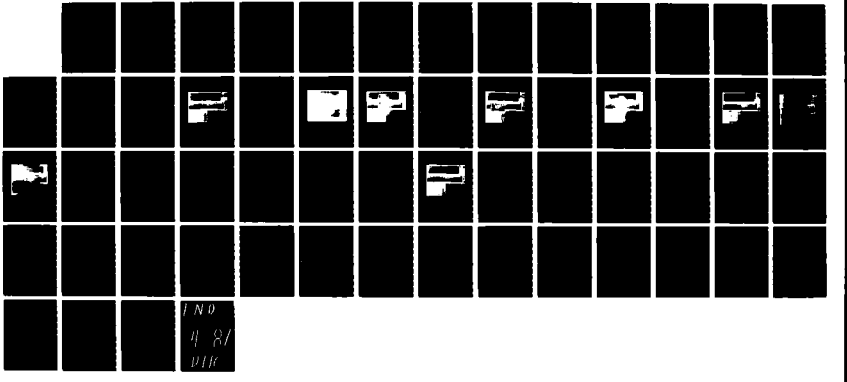
6) LOCATION. The LOCATION slot allows the pilot to specify the position value of an object. Although the FROM and TO slots were created, typically only the FROM position is of importance. An example of this is:

(Location (From OFF TO AIM

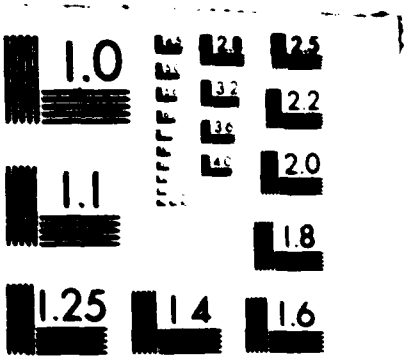
This slot specifies that the

AD-A177 717 PROTOTYPE KNOWLEDGE ACQUISITION AND REPRESENTATION EDITOR FOR THE F-16 FLIGHT DOMAIN(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. 2/2

UNCLASSIFIED D A SOBOTA DEC 86 AFIT/GCE/ENG/86D-4 F/G 9/2 NL



FND
4 87
DTIC




MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS

from the OFF position to the ARM positon. To ensure that the pilot enters a valid location for the current object, the editor provides a pop-up menu of defined loations for the pilot to choose from. For instance, if the pilot selected the MASTER-ARM-SWITCH as the current object, then the pop-up locations OFF, SIM, and ARM are provided to the pilot when he selects the "Enter Location" item in the editor. The pilot will be allowed to add or modify any of these location values. For modelling the CCIP, CCRP, and LADD scripts, the following objects and locations were defined:

* A-G Mode Button
    * Select
    * Deselect
* AG Submode OSB (Option Select Button)
    * STRF
    * CCIP
    * DTOS
    * EO-VIS
    * CCRP
    * LADD
    * EO-PRE
    * MAN
* Master Arm Switch
    * ARM
    * OFF
    * SIM

* Missile Step Button (rotary CCIP,DTOS,CCRP) (15)
    * DEPRESS
    * RELEASE
    * HOLD
* WPN REL (Weapon Release) Button
    * DEPRESS
    * RELEASE
    * HOLD
* Fire Control Radar (FCR) OSB
    * OFF
    * FCR "means radar is ON"
    * STBY "standby is achieved by depressing the STBY OSB on the MFD" (15)
* Weapon (Stores) OSB
    * SELECT
        * Release Options
            * SGL (single)
            * PAIR
        * Arming Options (Fuzing)
            * NOSE
            * TAIL
            * NSTL (nose-and-tail)
        * Number of releases "based on number of release pulses" (15)
        * Impact separation distance
    * DESELECT
* Target Pod (TGP) OSB

    * SELECT
    * DESELECT
* Throttle
    * POWER-BACK
    * PUSH-FORWARD
* Side-Stick
    * PULL-BACK
    * PULL-UP
* Trigger
    * SQUEEZE
    * RELEASE
* Steerpoint Rocker Switch
    * SP-1 (Steerpoint #1)
    * SP-5
    * SP-7
    * SP-25
* Radar Mode OSB
    * GM (ground map)
    * GMT (ground-moving target)
    * SEA ("as in water")
    * BCN (beacon)
    * AGR (air-to-ground ranging)
* Aiming OSB
    * IP (Initial Point)
    * TGT (Target)
    * OA1 (Offset Aimpoint #1)
    * OA2 (Offset Aimpoint #2)

7) FEEDBACK. The FEEDBSCK slot is used to specify actions that the pilot can request from the pilot associate computer. Varying levels of assistance from the computer, for the pilot, include:

* check
* advise
* monitor = check + advise.
* verify
* diagnose
* predict
* configure (put the data in a certain format)
* display

An example of the FEEDBACK slot is:

(Feedback (VERIFY))

The editor will not allow the pilot to define any other pilot associate actions, and currently no computer action is taken based on this slot.

8) CONSTRAINT. The CONSTRAINT slot is used by the pilot to specify to the computer that an action should not be allowed (it should be suppressed) or that the pilot would like the pilot associate to perform an action if the condition is met. Each constraint will contain an IF and THEN component. Each component is made up of three subparts: a noun, an relational operator, a logical symbol, and a value. In the following example, the nouns are NUMBER-OF-SELECTED-STORES-TYPE and MASTER-ARM-SWITCH, the relational operators are >

and =, the logical symbol in the THEN component is OR, and the values are 0, ARM, and SIM.

```
(constraint ((IF NUMBER-OF-SELECTED-STORES-TYPE > 0)
             (THEN (MASTER-ARM-SWITCH = ARM) OR
                   (MASTER-ARM-SWITCH = SIM))))
```

a) CONSTRAINT NOUNS. The nouns allowed in the constraint slot are the pilot-defined objects and the editor predefined aircraft default parameters. Before particular maneuvers can be performed, the current state of the aircraft and the HUD cues must be checked. The predefined aircraft parameters and their initial default values (the units are not entered as part of the data) are:

* altitude (10000 ft)
* distance-to-target (20000 ft)
* fuel-remaining (2000 lbs)
* heading (25 degrees)
* airspeed (1500 knots)
* vertical-velocity (1200 ft/sec)
* number-of-g-forces (2 g's)
* pitch-angle (5 degrees)
* time-to-go (0 seconds)
* time-delay (0 seconds)
* stores-type (MK-82)
* number-of-selected-stores-type (4 MK-82's)
* pullup-cue (True)
* breakaway-cue (False)
* inrange-cue (True)

* ripple-release (False)

* target-type (SA-7)

* surrounding-threat-type (SA-9)

b) CONSTRAINT OPERATORS. The operators allowed in the constraint slot are the following relational operators:

* less-than (<)

* greater-than (>)

* equal-to (=)

* less-than-or-equal-to (<=)

* greater-than-or-equal-to (>=)

c) CONSTRAINT LOGICALS. The logical operators allowed in the constraint slot provide the ability for the pilot to specifiy multiple constraint checking in the IF and THEN constraint case. The logical operators available to the pilot include:

* and

* or

* none-of-these (no logical desired)

Besides the IF and THEN part of a constraint, an optional UNTIL part is available. The UNTIL part can be used to apply a constraint for a period of time, or to prevent an action from occurring until the condition is met. Several automatic pop-up menus step the pilot through the building of the constraint. An example constraint with the UNTIL part is:

```
(constraint ((IF RIPPLE-RELEASE = TRUE)
             (THEN WEAPON-RELEASE-BUTTON = HOLD)
             (UNTIL NUMBER-OF-SELECTED-STORES-TYPE = 0)))
```

d) CONSTRAINT VALUES. The values allowed in the constraint slot include the locations defined for an object, or any value input by the pilot for the predefined aircraft parameters.

9) SIDE-EFFECT. The side-effect slot, as programmed, is used to set the value of a defined object or predefined aircraft parameter only, upon completion of a checklist item. An example side-effect slot is shown as:

```
(side-effect (MASTER-ARM-SWITCH = OFF))
```

10) Another function provided throught the pilot interface, assists in avoiding duplication of defined objects, verbs, panels, and all other slots. The editor keeps a separate list for each slots with its allowable instantiation values by the pilot. Furthermore, if the pilot attempts to create an already existing script or checklist subunit, the editor informs him of its existence and no action is taken. A graph of the knowledge base or a list of defined scripts and subunits is also available for viewing by the pilot. The editor dynamically maintains the lists and graph.

13) The editor provides a facility for renaming and/or deleting scripts, objects, panels, etc. In doing so, the affected items in the knowledge base are updated with the new name or deleted, as requested. In either case, the

affected checklist items are displayed to the pilot.

Computer's Frame Representation of a Script

The pilot interface is used to enter F-16 flight domain information necessary to represent script checklists so that an internal computer frame representation of the script can be generated. The interface to the expert was designed such that the user need not know how to program a computer. All he needs to know is the required information requested through the user menus concerning his domain of expertise. The frame representation conversion process is begun when the pilot selects the item off the pop-up menu called "Generate Internal Frame Format." After the pilot answers the editor question "Which script would you like to format?," the internal computer frame representation is generated for the specified script. This formatted frame representation is output to a file. The editor functions will be described in more detail in Chapter V.

An example of the computer's internal frame representation of the CCIP script, originally developed in Chapter III, based on additional input from the domain expert Major Frank is as follows:

```
(script (CCIP))
(constraint NIL)
(action (NIL))
(object (panel NIL) (button NIL))
(location (from NIL) (to NIL))
(feedback (NIL))
(actor (NIL))
(side-effect NIL)

(subunit (CCIP-1))
```

```
(constraint NIL)
(action (DEPRESS))
(object (panel ICP) (button A-G-MODE-BUTTON))
(location (from DESELECT) (to SELECT))
(feedback (NIL))
(actor (PILOT))
(side-effect NIL)

(subunit (CCIP-5))
(constraint NIL)
(action (SELECT))
(object (panel MFD) (button WPN-OSB))
(location (from DESELECT) (to SELECT))
(feedback (VERIFY))
(actor (PILOT))
(side-effect NIL)

(subunit (CCIP-2))
(constraint NIL)
(action (SELECT))
(object (panel MFD) (button AG-SUBMODE-OSB))
(location (from NIL) (to CCIP))
(feedback (VERIFY))
(actor (PILOT))
(side-effect NIL)

(subunit (CCIP-6))
(constraint NIL)
(action (SET))
(object (panel LEFT-MISC) (button MASTER-ARM-SWITCH))
(location (from OFF) (to ARM))
(feedback (VERIFY))
(actor (PILOT))
(side-effect NIL)

(subunit (CCIP-3))
(constraint NIL)
(action (DEPRESS))
(object (panel HANDS-ON) (button WEAPON-RELEASE-BUTTON))
(location (from RELEASE) (to DEPRESS))
(feedback (MONITOR))
(actor (PILOT))
(side-effect NIL)

(subunit (CCIP-4))
(constraint ((IF RIPPLE-RELEASE = TRUE)
             (THEN WEAPON-RELEASE-BUTTON = HOLD)
             (UNTIL NUMBER-OF-SELECTED-STORES-TYPE = 0)))
(action (RELEASE))
(object (panel HANDS-ON) (button WEAPON-RELEASE-BUTTON))
(location (from NIL) (to RELEASE))
(feedback (VERIFY))
(actor (PILOT))
(side-effect (MASTER-ARM-SWITCH = OFF))
```

A one-to-one comparison between the Chapter III rule-based description of the CCIP checklist and the final frame-based representation yields a better understanding of the above representation. After much research, thought, and discussion with Major Cross and Major Frank, the constraints dealing with HUD visual cues would not be included. The constraints were defined in terms of items an F-4 pilot would ask or expect of his backseater. Additionally, the order of subunits in the output file reflects the value of the PTR-ORDER slot for each subunit. As a result, even though CCIP-5 was the fifth checklist step added to the system, the pilot really wants it in the second position where it really belongs.

LISP Code Generator

Automated tools provide many useful functions. For example, the knowledge acquisition editor maintains the consistency of the knowledge base. A dependable and easy-to-use tool can be extended to provide additional desired user functions. One such function demonstrating the editor's extension to an application area is transformation of the internal frame format into a format executable on a computer. This transformation process is called "code generation."

The "translator," another name for the code generator, takes the internal frame representation of a script and converts it into a syntactically correct LISP program. At a

simplistic level, every time the word "if" is encountered, the LISP equivalent is "cond," while assignment of a slot value is represented by the "setq" command in LISP. The LISP code equivalent script is then executed on a LISP machine. This process could also be demonstrated for Ada code or another person's desired representation. Some day the generated code may be directly executed on the aircraft's computer system.

Because no simulator exists to execute the LISP code on at this time, a dummy routine called system-sensor-check pretends to check and update the aircraft settings. The seven parameters expected by this function are: the feedback, the actor, the panel, the object, the action, the from-location, and the to-location. The ordering of the parameters, as shown, is imperative.

The output of the LISP code generator module using the CCIP-script's internal representation, with the clear screen commands stripped out, appears as:

```
(defun CCIP ()
  (system-sensor-check (quote nil) (quote nil) (quote nil)
    (quote nil) (quote nil) (quote nil) (quote nil))
  (system-sensor-check (quote nil) (quote pilot) (quote icp)
    (quote a-g-mode-button) (quote depress) (quote deselect)
    (quote select))
  (system-sensor-check (quote verify) (quote pilot) (quote
    mfd) (quote wpn-osb) (quote select) (quote deselect)
    (quote select))
  (system-sensor-check (quote verify) (quote pilot) (quote
    mfd) (quote ag-submode-osb) (quote select) (quote nil)
    (quote ccip))
  (system-sensor-check (quote verify) (quote pilot) (quote
    left-misc) (quote master-arm-switch) (quote set) (quote
    off) (quote arm))
  (system-sensor-check (quote monitor) (quote pilot) (quote
    hands-on) (quote weapon-release-button) (quote depress)
```

```
    (quote release) (quote depress))
  (cond ((equal ripple-release (quote t))
         (loop until (equal number-of-selected-stores-type
                            (quote 0)))
       (setq weapon-release-button (quote hold))))
  (system-sensor-check (quote verify) (quote pilot) (quote
    hands-on) (quote weapon-release-button) (quote release)
    (quote nil) (quote release))
  (setq master-arm-switch (quote off))
)
```

Summary of Detailed Design

This chapter discussed the detailed design of the three modules within the knowledge acquisition and representation editor as applied in the F-16 flight domain. The three main modules comprising the editor are the pilot interface, the computer's rule-based frame representation format of the pilot's knowledge, and the LISP language code generator. Most of the details provided in this chapter were provided after implementing the prototype knowledge editor. Appendix A and B contains the equivalent frame representation and generated LISP code for the CCRP checklist, while the LADD output is not shown since it is almost an exact duplicate of the CCRP output.

# V. System Description and Analysis

## Introduction

This chapter first discusses the knowledge editor in general terms, followed by a specific description of each of its six activeimage boxes. The activeimage boxes represent actions performed upon the knowledge, including creating, editing, listing, viewing, and representing the knowledge, as well as a box for saving or disposing the knowledge upon exiting the editor. Following the system description is a brief description on how the menu driven system works. The final system information includes instructions on how to load the editor and execute the application specific script results. This chapter concludes with a subjective analysis of the knowledge editor by three fighter pilots and one attack pilot.

## General Editor Information

All knowledge in this editor is represented in frame format with the assistance of KEE's (Knowledge Engineering Environment), Version 2.1, built-in frame-based representation hierarchy structure. KEE, an expert system building tool (software package) developed by Intellicorp, provides a powerful, rapid prototyping environment on a LISP machine. The prototype knowledge editor was implemented on a Symbolics 3670 LISP machine. Currently, the editor does not make use of all of KEE's built-in features, such as the forward or backward chaining mechanism and the rule editor;

however, the object-oriented activeimages, cascading menus, and message passing facilities are extensively utilized. If the editor is expanded upon, so that the knowledge is represented in rules, and not just scripts, then the forward and backward chaining mechanism will become crucial.

For the flight domain application, each checklist item in a script represents a "subunit" in the knowledge editor. To assist the pilot in specifying information about a newly created subunit, pre-defined subunit frame slots are created and displayed in the KEE output window with values of "NIL" (meaning nothing). The slots are instantiated with values through the editor CREATE activeimage subfunctions. The specific frame slots fully specifying a subunit checklist item are: action, object, location, panel, feedback, actor, constraint, side-effect, and its specific position number (ptr order) in the script. As discussed in Chapter IV, the location frame includes two parts, the FROM and the TO positions, while the panel slot is a subpart of the object slot.

Feedback to and input from the pilot about a particular action is provided through the lisp listener window on the screen. Special care was taken to ensure the pilot knows when an action is complete. For instance, the message "O.K." is printed to the screen in the lisp listener window at the end of every action.

System Functions

The knowledge editor has six simple value display

'activeimage' boxes used for activating a different pop-up menu associated with each box. These editor functions are CREATE, EDIT, LIST, VIEW, OUTPUT-REP, and EXIT. Some of the subfunctions available within each of the six boxes will now be discussed in more detail.

CREATE Function Box.

Within the CREATE activeimage, the pop-up menu attached to it allows for eleven primary actions, each with zero to four suboptions. Once the pilot specifies whether a new script or new subunit will be created, the particular information for that subunit can be supplied using the remaining nine primary menu actions. For the subunit, the additional information allowed is the position order, action, object, panel, locations, actor, feedback, constraint, and side-effect. For each submenu, except the position ordering, an arrow exists to the right of the pop-up menu selection. As an example of submenus, see Figure 23. The arrow implies that submenus associated with that menu item exist. To display these pop-up menu subitems, the mouse symbol must be moved to the right, through the arrow.

The main menu attached to the CREATE box is activated when the pilot puts the mouse in this box, and clicks the the mouse left button. A submenu is displayed when the pilot moves mouse symbol to the right through the arrow after the menu item "Enter Script Subunit Name." The pilot can now choose to look at an example subunit name or to define a new

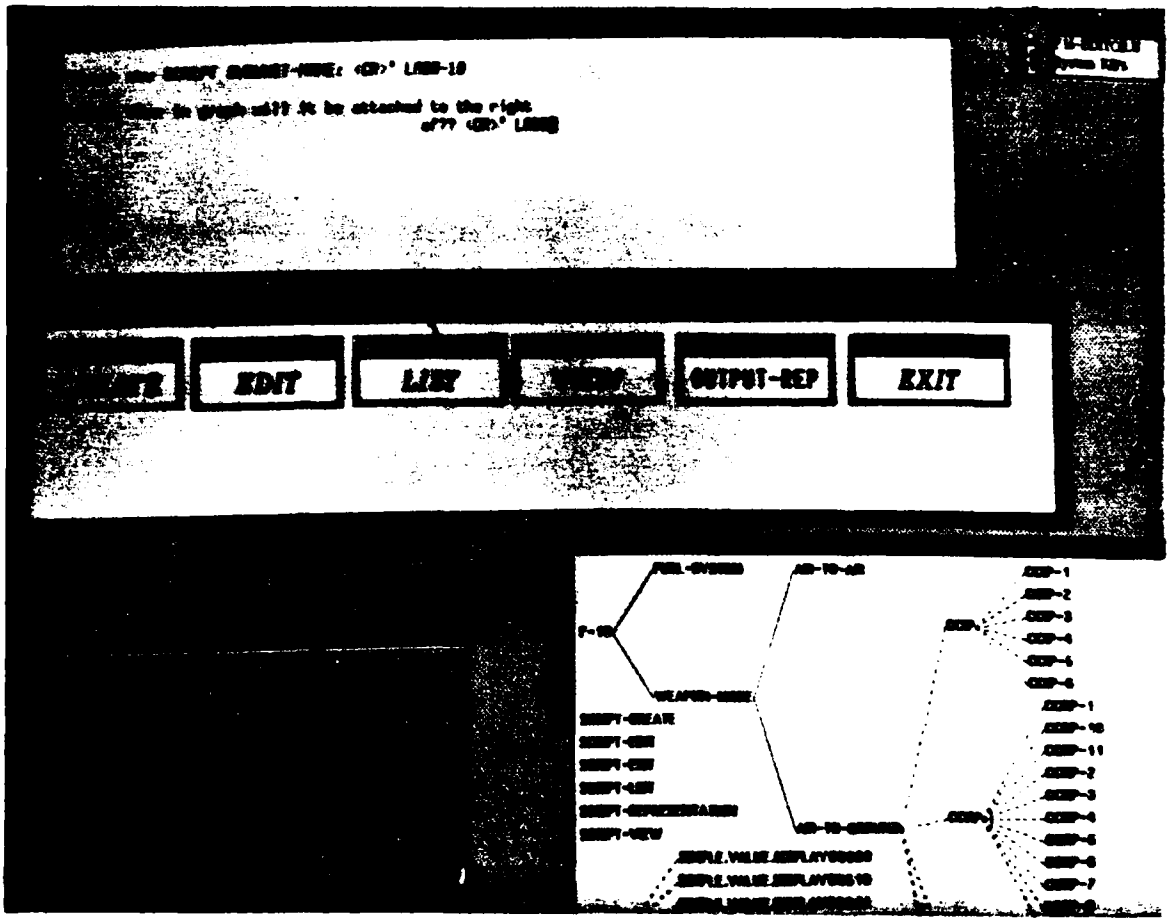



Figure 23. Result of Selecting CREATE's "Enter Script Subunit Name"

script subunit.

EDIT Function Box.

The EDIT activeimage box is very similar to CREATE. It has the same eleven primary actions available from its main pop-up menu. However, the subfunctions are different. The pilot selects this box for modifying a slot or slots for an existing subunit. The pilot cannot create any new objects, locations, and so forth, within the edit box, unlike that of the CREATE box; however,the pilot can view example values in both the EDIT and the CREATE box.

The EDIT activeimage provides the pilot the opportunity to rename or to delete an object, panel, location, subunit, or action. It also provides the pilot the capability to change a slot value or to reset it back to its default NIL value is provided.

For instance, if the pilot displays the submenu items of "Modify Script Object," the options shown are "Show Example Object," "Select a Defined Object," "Rename a Object," Delete a Object," and "Reset Value of Object Subunit" (see Figure 24). Now if the pilot choses the item "Select a Defined Object" by clicking the mouse left button, a large font selectable pop-up menu of all the defined objects in the F-16 flight domain are displayed on the screen (see Figure 25). The pilot selects the desired object by positioning the mouse symbol over that object's name, then by clicking the mouse left button. The updated value will appear in the current subunit's object slot shown in the KEE

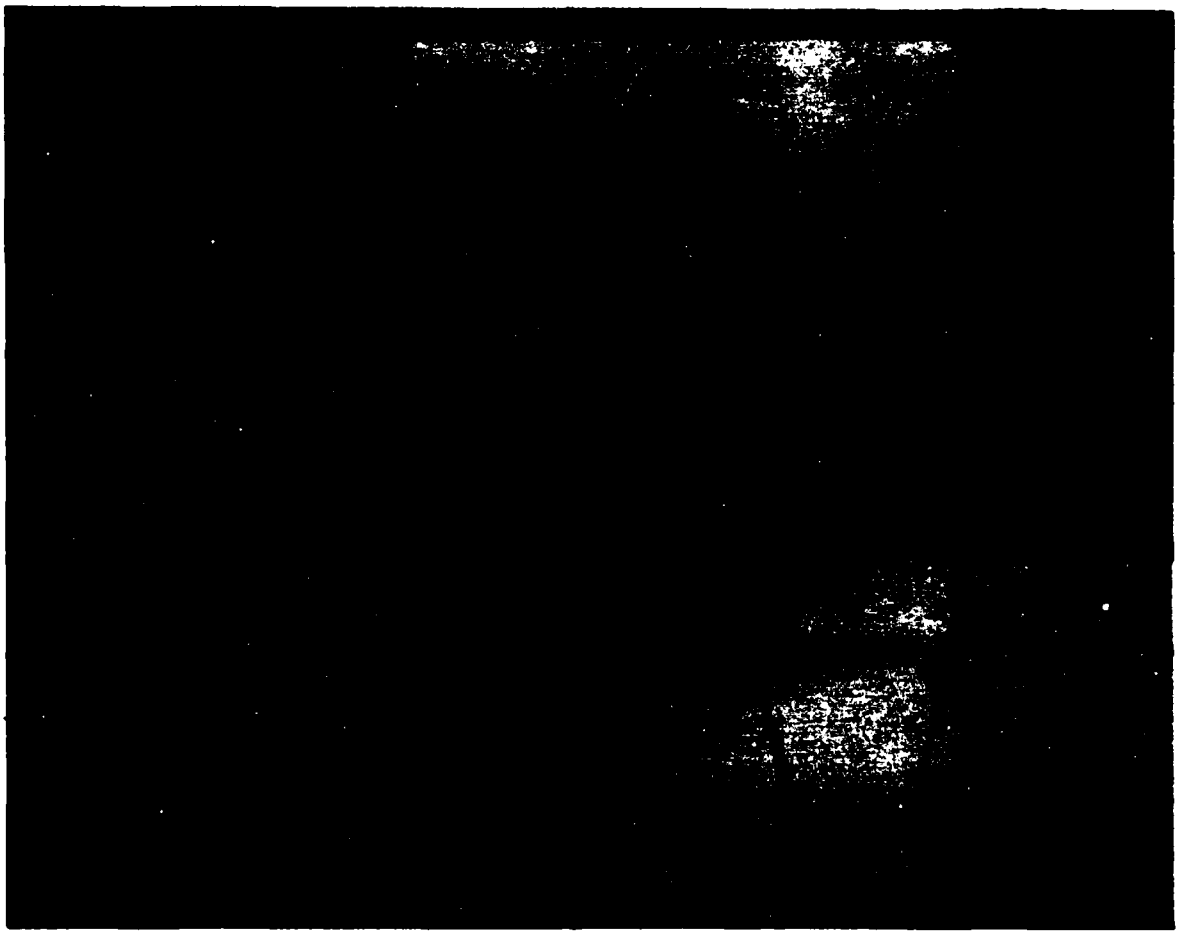


Figure 24. Submenu for EDIT's "Modify Script Object"

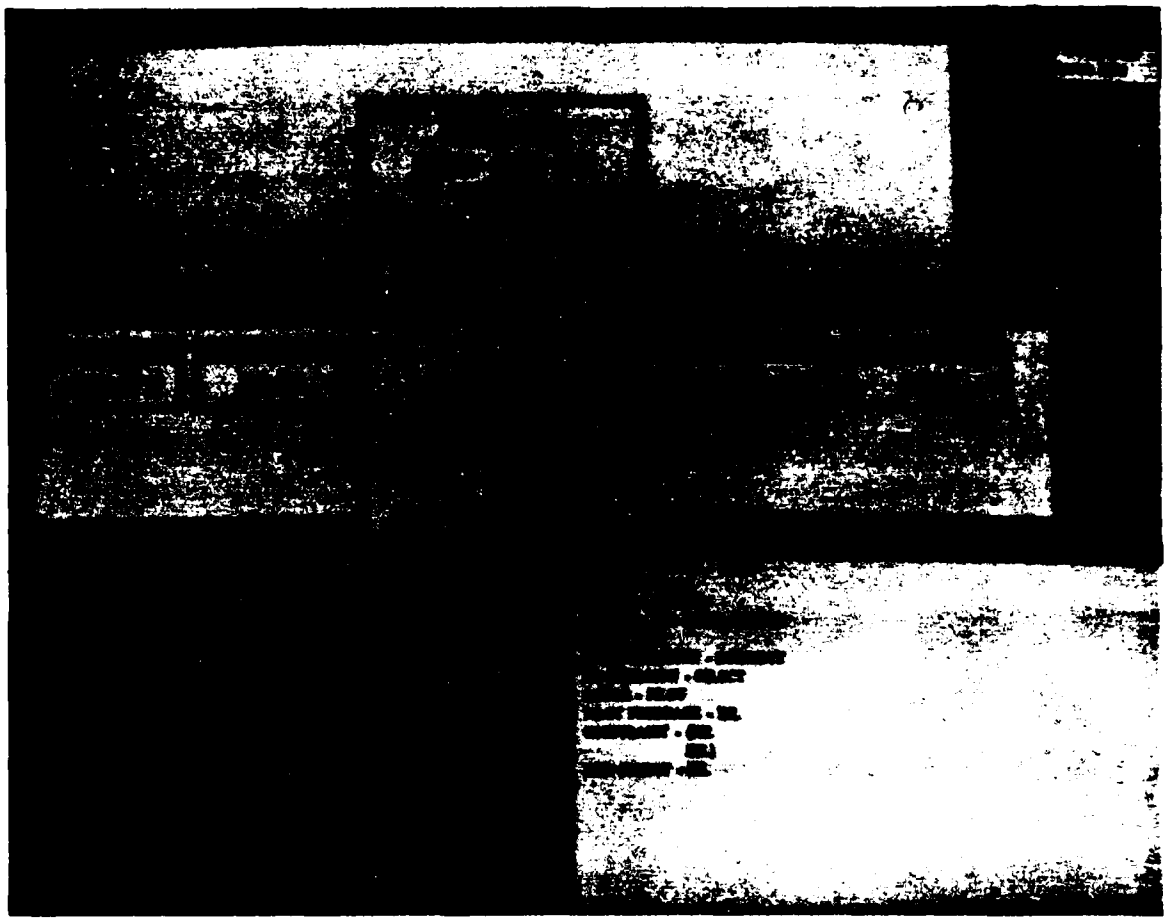


Figure 25. Result of Selecting EDIT's "Select a Defined Object"

output window.

LIST Function Box.

Within the LIST activeimage, the pop-up menu attached to that box allows for eight primary actions, none containing a submenu. This activeimage provides the pilot with an informational list of all the defined scripts, subunits, actions, objects, panels, locations, feedback actions, or a particular subunit's defined slots (see Figure 26). The pilot should not choose an item from the pop-up menu, but if one is purposely or accidently chosen, the editor takes no action. This box provides the pilot with passive informational actions only.

After representing the CCIP and CCRP air-to-ground weapon delivery modes in this editor, the following objects are currently defined in the F-16 knowledge editor system: AIMING-OSB, RADAR-MODE-OSB, AG-SUBMODE-OSB, STEERPT-ROCKER-SWITCH, TGP-OSB, FCR-OSB, WPN-OSB, TRIGGER, THROTTLE, SIDE-STICK, A-G-MODE-BUTTON, WEAPON-RELEASE-BUTTON, MISSILE-STEP-BUTTON, and the MASTER-ARM-SWITCH; where OSB stands for Option Select Button (one of the 20 buttons around the MFD (Multifunction Display), TGP stands for Target Pod, AG stands for Air-to-Ground, STEERPT stands for steerpoint, and FCR stands for Fire Control Radar.

Each object has associated with it valid pilot-defined locations (positions) that they may assume. For example, the MASTER-ARM-SWITCH can take on the one of three

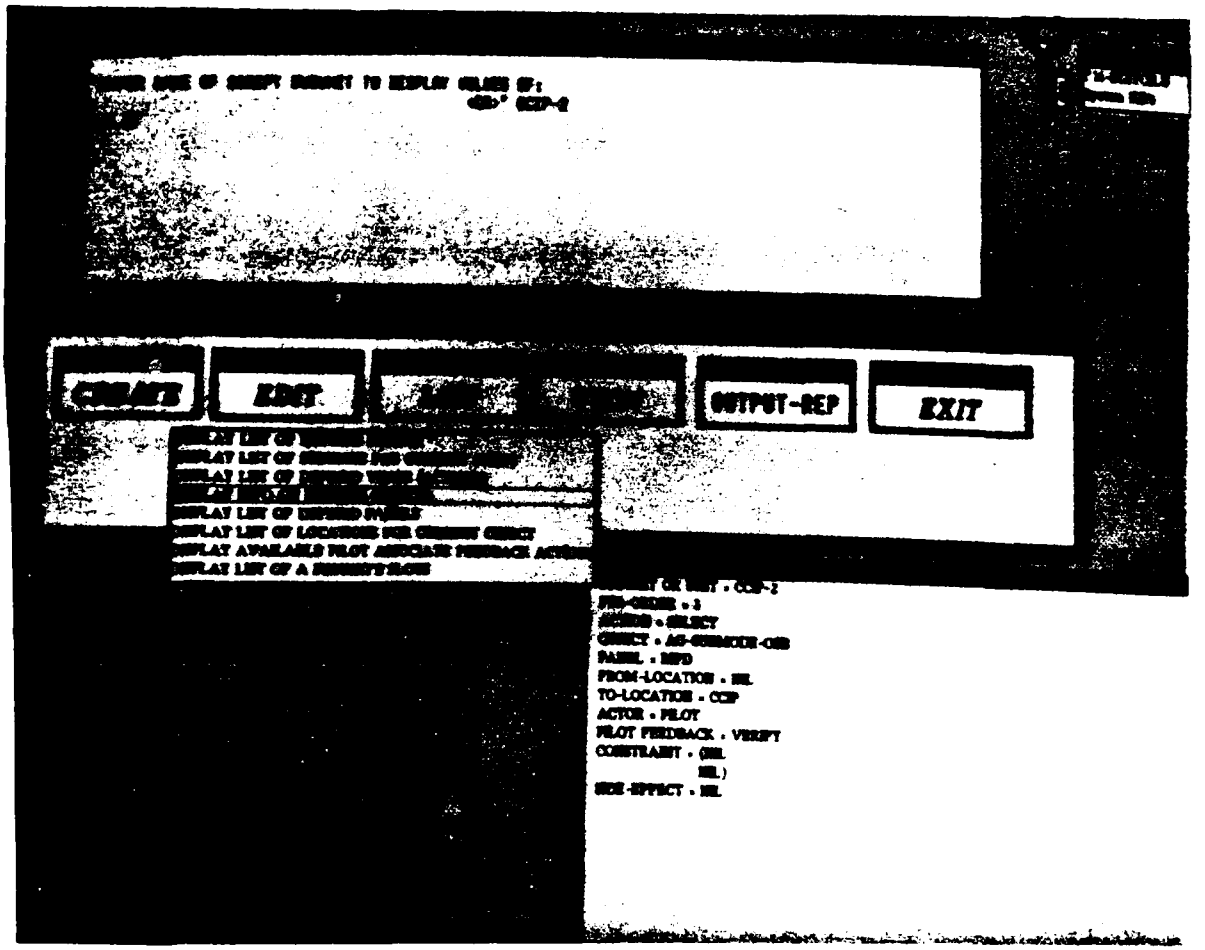



Figure 26. Menu for LIST box

locations: OFF, SIM, and ARM. However, the trigger has only two defined locations: SQUEEZE and RELEASE. The actions available on all generic objects, thus far, are MOVE, SELECT, SET, DEPRESS, and RELEASE. Lastly, the existing panels consist of the UPFRONT-CONTROL, HANDS-ON, FIRE-CONTROL-PANEL, DED (Data Entry Display), MFD, LEFT-MISC, and ICP (Integrated Control Panel).

VIEW Function Box.

Within the VIEW activeimage, the attached pop-up menu allows for five primary pilot actions, none containing a submenu. All five actions affect the display in the KEE window labelled OUTPUT located in the lower right hand corner of the display. The three most frequently used functions are a text display of a subunit's slot values only (see Figure 26 again), a graphical display of all the defined F-16 scripts and their subunits (see Figure 24 again), and finally, a text display of the current aircraft parameter values (see Figure 27). These functions are activated when the pilot selects the pop-up commands, "View a Subunit's Slot Values Only," "Graph the Entire Knowledge Base," and "View Current Aircraft Constraint Values," respectively.

OUTPUT-REPRESENTATION Function Box.

Within the OUTPUT-REP activeimage, the pop-up menu attached to it allows for three pilot actions, one of which is only a dummy stub for English sentence generation. The

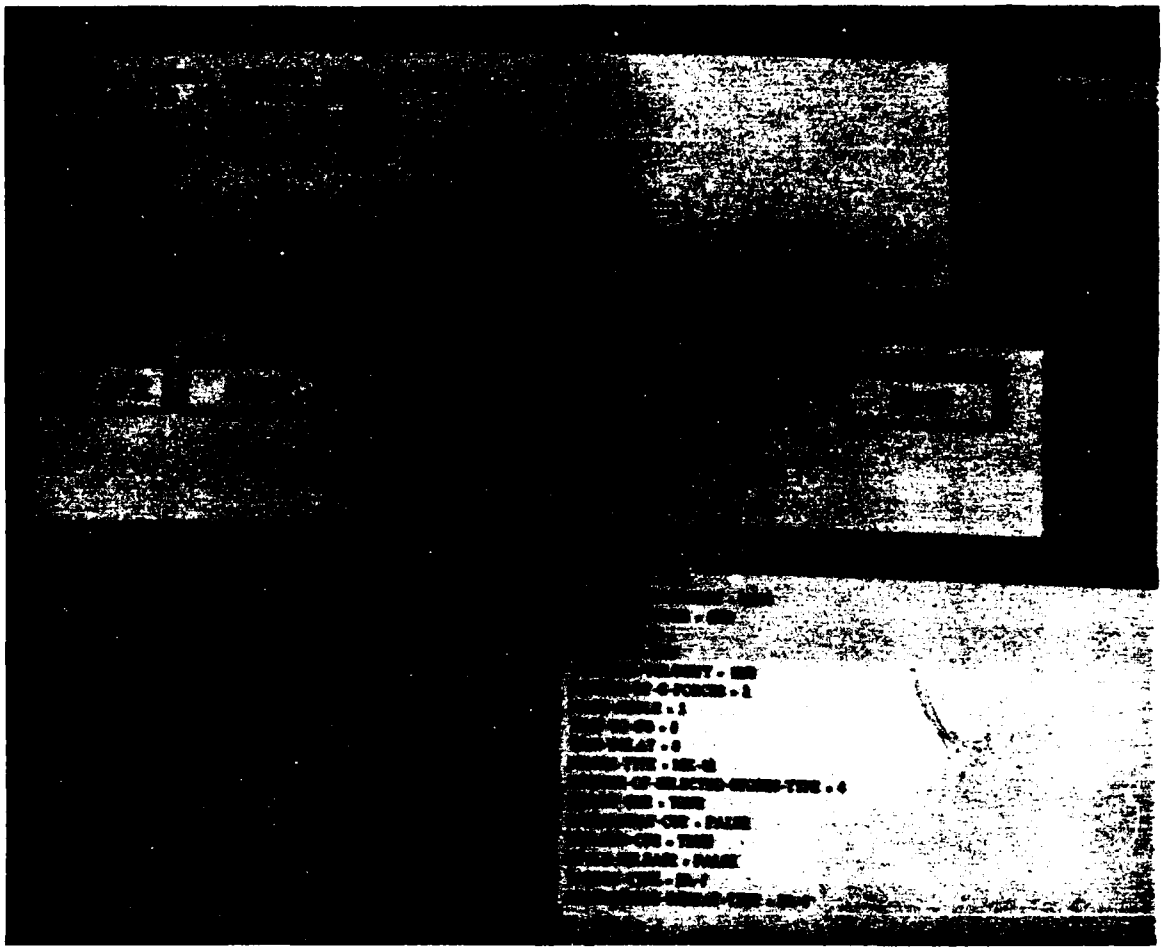


Figure 27. Result of Selecting VIEW's "View Current Aircraft Constraint Values"

two active functions comprise two of three primary editor features shown in Chapter III, Figure 13. The computer internal frame representation is generated when the pilot selects the menu item "Generate Internal Frame Format." The pilot is prompted for the script name, whereupon a file containing this frame format is generated with the desired script name. The frame format may then be used as input in generating the equivalent LISP code using the command "Generate Lisp Language Code" (see Figure 28). Again, the editor will prompt the pilot for the name of the script to use as input into the LISP code generator module. The results of the two actions can be found in files with the name of the script and the extension .FRAME or .LISP, respectively.

A portion of the internal frame format for the CCIP weapon delivery mode is shown in Figure 29. Each script unit and its subunits contain the same repeating groups of slots, which may contain a NIL value.

Two important points about the services provided by the editor in generating the frame format: 1) the pilot did not have to worry about properly arranging the parentheses, or grouping the data, and 2) most of the slot values were filled when the pilot selected items off a pop-up menu, thereby eliminating the chance of a spelling error.

Figure 30 shows the equivalent LISP code generated for the CCIP script. A LISP function called SYSTEM-SENSOR-CHECK was created to simulate the checking and updating of

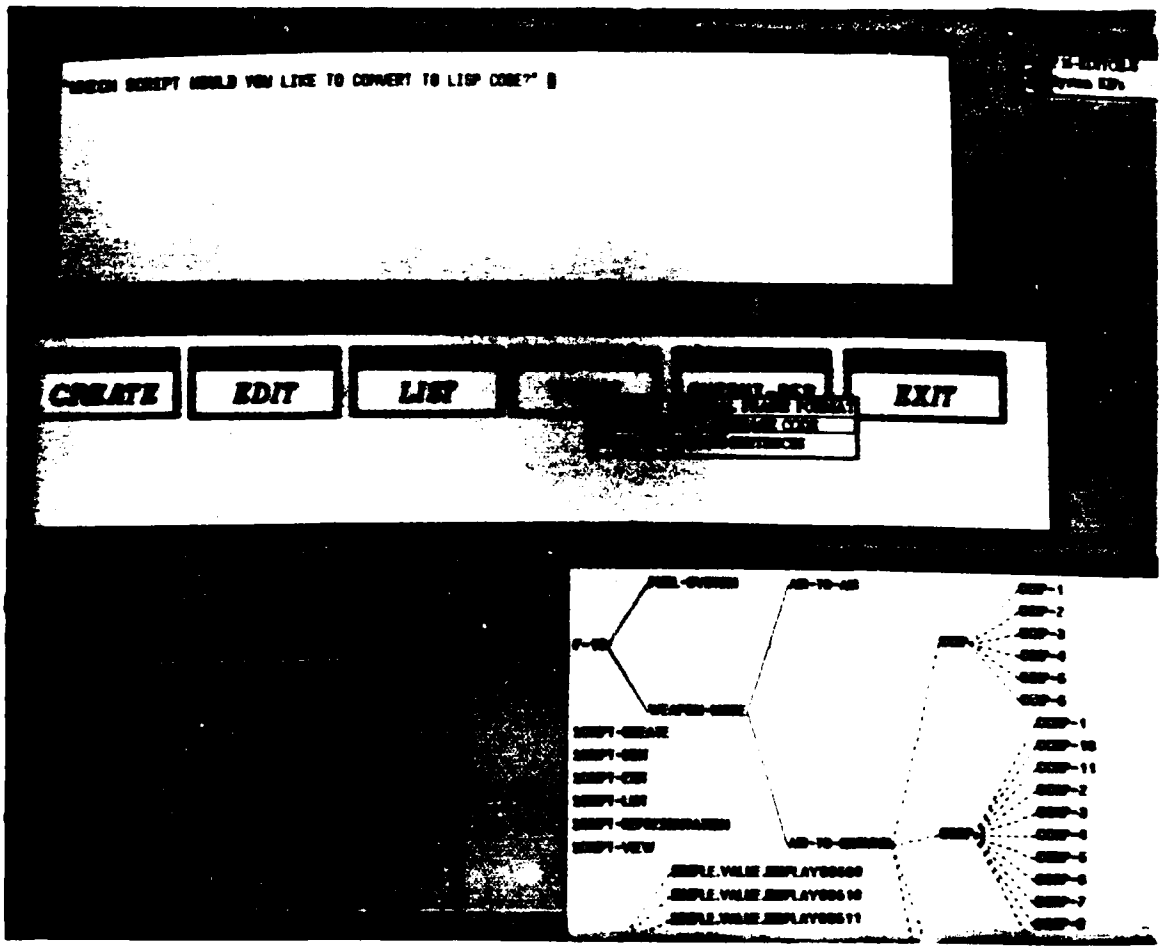



Figure 28. Menu for OUTPUT-REP box

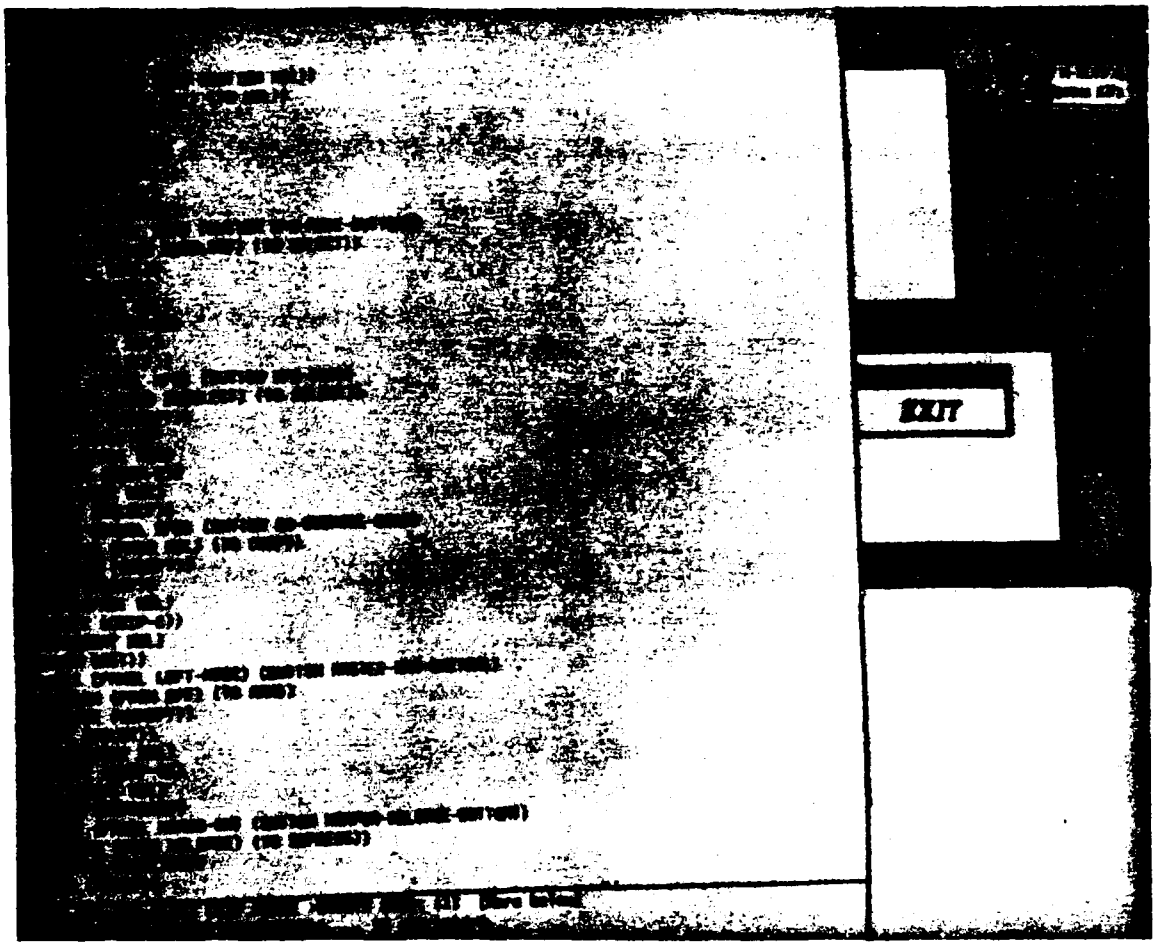



Figure 29. Sample FRAME FORMAT output file for CCIP script

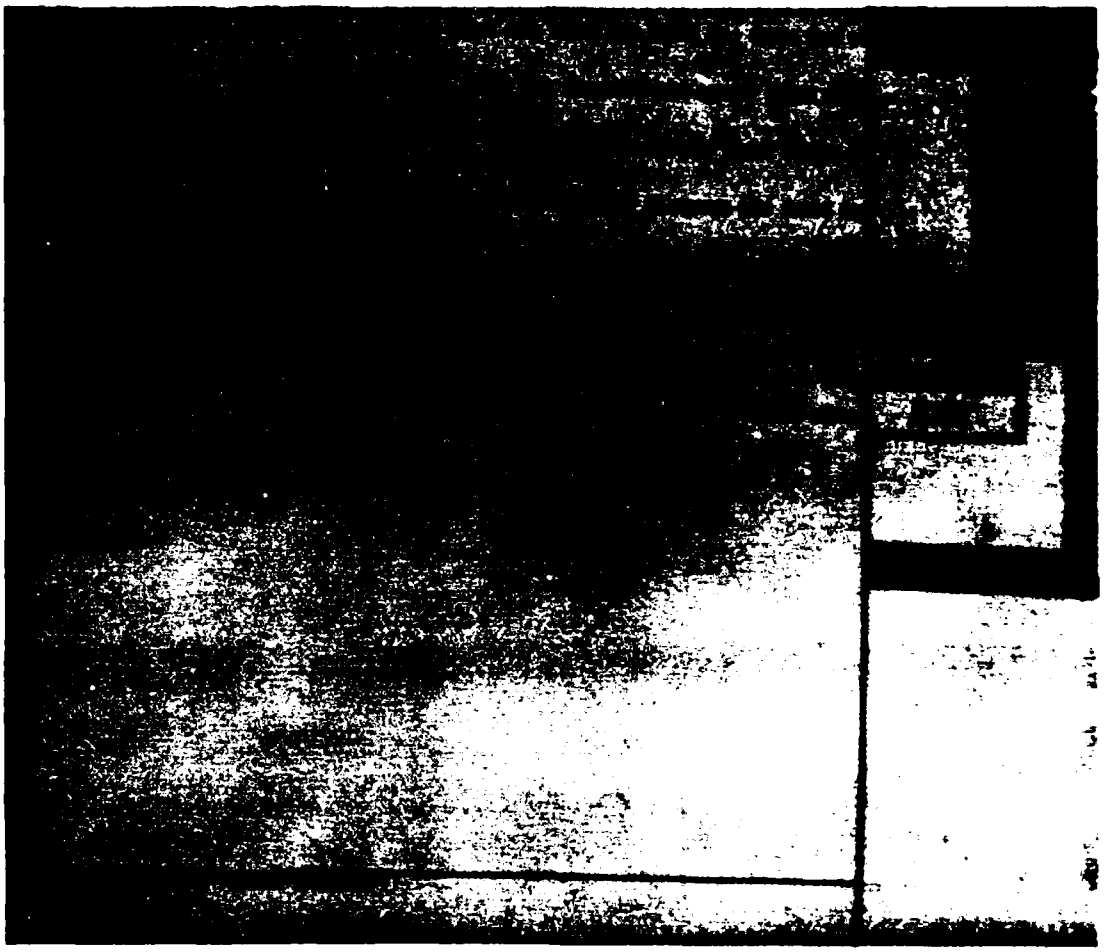


Figure 30. Sample LISP CODE output file for CCIP script

aircraft settings. This function accepts the slot values for each subunit, sorts out the information, and passes it to the appropriate computer or panel capable of checking the value of a switch, button, or sensor. Since no F-16 aircraft simulator is available at this time, the data is split between two functions based on the value of the actor slot, either computer or pilot. With no real sensors to check, there are no built in heuristics to decide if a checklist step was properly executed. In the meantime, a random number generator is used to decide whether a warning should be issued to the screen.

EXIT Function Box.

The EXIT activeimage presents the pilot with two actions. The first action, "Save All Changes," allows the pilot to save the modified KEE knowledge base and all of the files created by the editor containing the switch locations, new objects, actions, etc. When the knowledge files are saved, if a previous version existed, it is over-written with the new version. This was an implemenation decision made to avoid the occurrence of an enormous directory size. The second action, "Forget Changes & Reload Old Version," allows the pilot to flush the knowledge base editor of all data entered since the last time the "Save All Changes" option was selected. In both cases, the lisp listener window is used to provide feedback to the pilot on the status of the action.

How the Menus Work

Within each of the six activeimage boxes, CREATE, EDIT, LIST, VIEW, OUTPUT-REP, and EXIT, KEE provides the facility for handling mouse clicks through the use of its "own slots." For example, with the activeimage box labelled CREATE on the editor panel, KEE assigned it the name-number SIMPLE.VALUE.DISPLAY00509. The MOUSELEFTITEMS own slot of the activeimage contains three pieces of data for each menu item: 1) the information to display on the attached pop-up menu, 2) a keyword index to send to the MOUSELEFTBUTTONFN own slot of the same activeimage based on the menu item chosen, and 3) the pilot help documentation line displayed at the bottom of the LISP machine screen.

Instead of entering all of the LISP code necessary to service the menu items for the CREATE box in the MOUSE-LEFTBUTTONFN own slot of the SIMPLE.VALUE.DISPLAY00509 activeimage, the author chose to insert a call to a function that contains all the necessary code. This holds true for all of the activeimage boxes; however, each calls a differnet servicing function. All of the LISP code used to service the activeimage pop-up menus was grouped into a file called EDITOR.LISP under the SOBOTA directory. This is one of the files that the DRIVER.LISP file loads when bringing up the editor.

For an example of how this menu interaction works, the knowledge editor tool can be loaded as described in the section to follow entitled "Loading the Knowledge Editor,"

Once loaded, the user can put the mouse symbol in the CREATE box, followed by clicking the MIDDLE mouse button. When the attached system menu is displayed, the user moves the mouse through the arrow to the right of the word DISPLAY on the menu, then puts the mouse on the third item IMAGE UNIT and clicks left. The SIMPLE.VALUE.DISPLAY00509 unit should be displayed in the KEE window labelled OUTPUT. Now the user should move down to the own slot MOUSELEFTITEMS, and look at the contents of its VALUES attribute. The contents begin with "Enter Script (Name)" (Quote Name) "enter as one word, can have hyphens in it." The first part, in double quotes, will appear on the pop-up menu attached to the CREATE box. The second part, preceded by a single quote mark, is the word passed to the MOUSELEFTBUTTONFN own slot to be used an an index into the code. In this case the word NAME is sent. Looking at the value attribute of the MOUSELEFTBUTTONFN, a call to the function SCRIPT-CREATE-LEFTBUTTON is substituted for the actual LISP code. This function is held in the EDITOR.LISP file, as already mentioned. Below is a partial copy of the function SCRIPT-CREATE-LEFTBUTTON in the EDITOR.LISP file demonstrating how the keyword index works. Currently there are over 30 keyword indexes in this function, one for each menu and submenu item. If the keyword sent to this function is NAME, then the code after the word NAME is executed, until the next keyword EXAMPLE-NAME appears.

```
(defun script-create-leftbutton (thisunit &rest ignore)
   (let
      ((command (menu
        (get.slot.menu 'simple.value.display00509
                       'mouseleftitems))))
      (when command
        (selectq command
          (name (send lisp.listener :clear-window)
                (print "Enter new script name: <CR>"
                                lisp.listener)
                (setq *name* (read lisp.listener))
                (cond ((equal
                         (member *name* *script-name-list*)
                        nil)
                       (setq *script-name-list*
                         (cons *name* *script-name-list*))
                       (graph.kb 'f16-editor.u)
                       (print "which item in graph will it
                         be attached to the right of??
                         <CR>" lisp.listener)
                       (setq *parent*
                         (read lisp.listener))
                       (create.unit *name* 'f16-editor.u
                         nil *parent*)
                       (print *name* lisp.listener))
                      (t
                       (print "script already exists in
                         graph, no action taken"
                         lisp.listener)))
                (print "o.k." lisp.listener))
          (example-name (ALL ITS CODE))
          (new-name (ALL ITS CODE))
             .
             .
             .))))
```

Lastly, the third part of the MOUSELEFTITEMS slot contains the words appearing in double quotes, "enter as one word, can have hyphens in it." These exact words will appear in the LISP machine's documentation pilot help line at the bottom of the screen. As a final comment on creating menus, the submenus within the menus were created using the reserved word, SUBITEMS, as shown in the own slot MOUSELEFTITEMS. The user should be careful of parenthesis placement when the subitems are created, using the KEE

keyword SUBITEMS.

Loading the Knowledge Editor

To bring up the knowledge editor on the Flight Dynamics Laboratory Symbolics 3670 LISP machine, the following steps are necessary:

1. Boot up the KEE world. (for the Flight Dynamics Lab this entails the following)
   a. Type (si:halt) to lisp listener.
   b. At FEP> prompt, type b, then hit space bar. The system should complete the boot command, giving a default optional world to boot. If it says kee.boot, just hit RETURN key. Else type kee.boot, then hit RETURN key.
   c. Enter date and time, when requested.
   d. When system is booted complete, hit the SELECT key, then the letter 'K' key to switch to KEE.
2. Type (login 'sobota) in KEE lisp listener window.
3. Type (load "driver.lisp" 'kee) in KEE lisp listener window.
4. When requested, click the left mouse button to recreate the F-16 knowledge editor panel display.
5. When requested, re-position the KEE lisp listener window with 2 left mouse clicks to the desired size, following the instructions in the KEE lisp listener window. The recommended size is about 8" wide and 2.5 to 3" tall.

When the message : All ready, GO FOR IT!!!
GOOD LUCK!!

appears in the KEE lisp listener window, the knowledge editor is ready for use by the pilot. All of the necessary files were automatically loaded for the pilot from the DRIVER.LISP file.

Running the Results

When the pilot wishes to execute the code for a created script, he must first perform the required actions to create a .FRAME file, then a .LISP file as described under the

heading "OUTPUT-REPRESENTATION Function Box." To execute a script, the .LISP file for that script must first be loaded into the KEE environment (package). Loading the CCIP checklist script requires the following command typed into the KEE lisp listener:

(load "ccip.lisp" 'kee)

After the LISP machine replies that the file is loaded with some cryptic pathname, the script can be executed in the same format as any LISP function; simply type the name of the function into the KEE lisp listener window as shown:

(ccip)

The results of executing this script are shown in Figure 31. Obviously when the script is executed in a simulator or in the actual flight domain, the pilot only wants to know when a checklist item was incorrectly performed or not performed at all. He cannot be bothered with extraneous, unimportant information. Hopefully, someday a complementary relationship between the computer and the pilot will allow the computer to handle warnings without disturbing the pilot's deep concentration, thus resulting in an even higher level of situation awareness.

Analysis of Knowledge Editor

A subjective analysis of the knowledge acquisition and representation editor was performed by Maj. Dick Frank (F-16 pilot), Lt. Col. Ron Morishige (F-4 pilot), Maj. Ron Van der Weert (F-15 pilot), and Norm Geddes (A-7D pilot).

Using the rapid-prototyping approach for development of

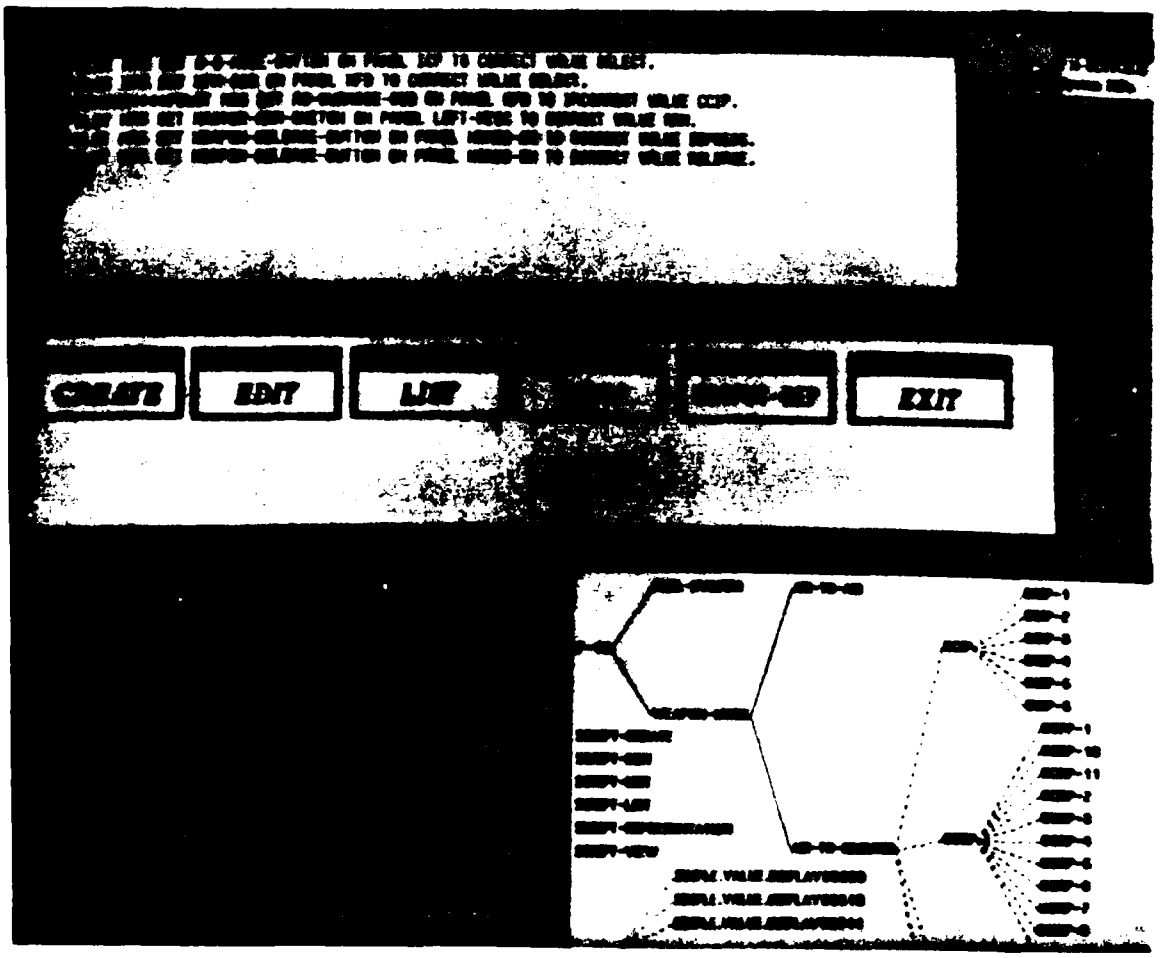



Figure 31. Results of Executing CCIP Lisp Code

the editor yielded a more useful product. The input from the pilots during design and implementation of the prototype knowledge editor produced a more accurate design, reflecting the features necessary for a knowledge acquisition and representation editor. Additional features, such as reordering of checklist components based on a slot called ptr-order, as well as the renaming and deleting of scripts, checklist items, objects, panels, actions, and locations were added as a result of the prototype implementation. Furthermore, the interactive method of constraint and side-effect building was provided after initial prototype testing on the CCIP and CCRP test cases.

The pilots agreed that a more flexible method of ordering script checklist components was still needed in future editors. If one checklist item's order in the script is changed it may affect several others in the script, so they too must be changed. Part of the problem could be corrected by writing an insert function that puts the subunit in the correct order, thus modifying the other affected subunits. However, a method for specifying that the order of some checklist items can be switched, yet others definitely must occur prior to another one occurring is still desired. For example, maybe checklist item #1 must occur first, and items #2, 3, and 4 can occur in any order, but all 4 items must definitely be completed prior to item #5 in the checklist. This example is a more realistic situation for the flight domain.

Another pilot comment concerned the fact that only one constraint per script checklist item is allowed. Although, the constraint itself allows for two parts, AND and OR, the accurate reflection of the flight domain will require additional constraints. A solution may be to query the pilot on the number of constraints he will be entering, so the current code could be looped the appropriate number of times; however, care must be taken by the next software developer when stripping apart the different constraints for conversion into the frame-based internal computer format. Also, as implemented the editor does not allow the pilot to put information in the main script frame slots. This might be a requirement in the future, particularly for the constraint slot.

Geddes felt that the utilities provided for consistency checking and avoidance of spelling errors for knowledge entry will be required by all expert system development tools. He said the constraint modelling in the editor reflected more of a side-effect flavor. Geddes thought constraint checking should only return the value true or false, not change or prevent the pilot from changing any switches in the aircraft.

According to Maj. Frank, explicit frame-based representation is a difficult concept to master; however, the editor is built so that the pilot need not know what a frame is or any other internal implementation details, for that matter. The pilot does not have to master the frame-

based concept while entering knowledge into the editor for the computer to know exactly what is expected of the pilot. Maj. Frank also believes that the integration of the pilot associate modules with existing aircraft subsystems will be a challenging task.

Overall, the prototype knowledge acquisition and representation editor more accurately reflects the features required by the pilot, due to the rapid prototyping approach. Many critical suggestions for a final product resulted from the prototype implementation, due to pilot involvement and feedback. The product editor performs the functions of maintaining the consistency of the knowledge base, providing mouse-sensitive pop-up menus for knowledge entry, providing no possibility of unmatched items during chaining due to a spelling error, providing no possibility of syntax formatting errors, and an environment for continuously expanding the knowledge base.

Summary of Knowledge Editor System

An overview of the knowledge editor, followed by a specific description of each of its six activeimage boxes was presented. The activeimage boxes represent actions performed upon the knowledge by the pilot, including creating, editing, listing, viewing, and representing the knowledge, as well as a box for saving or disposing the knowledge upon exiting the editor. Following the system description was a brief description on how the menu driven

system works. The final system information included instructions on how to load the editor and execute the application specific scripts. This chapter concluded with a subjective analysis of the knowledge editor by four pilots.

# VI. Conclusions and Recommendations

## Conclusions

The prototype knowledge editor provides many useful features for knowledge acquisition and representation. Simple errors now hindering the development of large expert systems, such as misspelled entries and incorrect formatting of the knowledge are eliminated through the use of the prototype editor tool built during this thesis research.

With such an easy-to-use, primarily mouse-driven tool, the knowledge engineer should have no trouble involving the expert in the representation of his knowledge about the flight domain. Direct expert involvement and a consistency checking editor should do much to cure the present problems in the rule acquisition process for expert system development. Such an interactive tool for constructing knowledge bases might also be used as an instrument for studying how learning is accomplished (9).

## Recommendations for Future Work

1) Captain Knode argues that plan-based processing is needed to fully model the complex flight domain (23). Script-based processing is not dynamic enough. As a result, the knowledge editor developed in this thesis should be expanded to acquire and represent plan-based knowledge. Knode supported the use of the Plan Applier Mechanism (PAM) as the chaining mechanism for plans during simulation.

2) The KREME editor being developed by BBN Laboratories

should be analyzed as a knowledge editor upon which to build a pilot knowledge acquisition interface.

3) The knowledge editor could be integrated with an Ada environment, so that the pilot could define specifications and the editor directly generate Ada code for execution on future avionics systems. The Flight Dynamics Laboratory recently (in October 1986) purchased and installed an Ada compiler on its Symbolics 3670 LISP Machine in AFWAL/FIGL, the same machine used for development of this prototype knowledge editor.

4) Using either the KREME editor or the knowledge editor developed for this thesis as a starting point, the research could extend the editor to acquire and represent information about the Pilot's Associate Mission Planning module. The concurrent thesis research of 2Lt. Jeffrey S. Bradshaw (AFIT, GCS-86D) entitled "A Pilot's Aid for Route Selection and Threat Analysis in a Tactical Environment" provides interactive route selection, rudimentary threat analysis (what threat threatens what leg and for how long), and building of a flight card for the mission, based upon the selected route. Furthermore, threats can be suppressed or defended against, and the current intelligence state can be displayed. Bradshaw's research models the F-16 aircraft, but can be easily changed for any other aircraft. It could be used as a model of how mission planning is accomplished. Integration of the knowledge editor and Bradshaw's work

offers much potential.

5) Another concurrent research effort, Capt. Wesley M. Smith's (AFIT, GCE-86D), entitled "An Artificial Intelligence Aid for B-1B Mission Replanning" finds a path through ten different types of threats, given a new target. The main constraint used by Smith is time over the target. Using the knowledge editor to enter in the pilot's expertise, threat information, and target information required by Smith's prototype replanner provides for an application proof of concept.

6) A third concurrent thesis by Capt. Arthur L. Estes (AFIT, GCS-86D) entitled "Knowledge-Based Automatic Test Program Generation for the Abbreviated Test Language for All Systems" provides generation of ATLAS (Abbreviated Test Language for All Systems) source code, given test specifications input by the user. This research was performed for Warner Robbins Air Logistics Center. This research further demonstrates the value of automatic software tools.

The remaining suggestions are small extensions of the knowledge editor developed under this research.

7) For efficiency's sake, the popup-menu code could be broken out into its own routine. Two parameters would be needed, the list to be displayed and the string representing the title (header) at its top.

8) A slot called STORAGE was anticipated by the author for specifying whether a script, plan, goal, action, or subgoal was being defined in the knowledge editor. This thesis addresses only the script aspect of understanding, therefore the storage slot was not needed; however, it is now mentioned for future consideration during expansion of the editor. Following Wilensky's approach in McPAM, as pointed out in Chapter II, if the user desires to enter a rule, he must specify whether he intends for it to perform instantiation, planfor, subgoal, or initiate. Once specified, the correct if/then template should appear for instantiation.

The script format is: (Storage (Script)).

The rule formats include:

| | |
|---|---|
| Instantiations are : | |
| if plan then action | (Storage (Action)) |
| Planfors are: | |
| if goal then plan | (Storage (Plan)) |
| Subgoals are: | |
| if plan then goal , or | |
| if goal then goal | (Storage (Goal)) |
| Initiates are: | |
| if theme then goal. | (Storage (Theme)) |

The computer applies these planning rules during flight by sensing the pilot's actions, interpreting them, and determining the pilot's goal (what he is trying to accomplish).

9) The additional pre-defined data accessor functions for modification of the knowledge base when it reaches simulator

testing phase and when it includes representations other than scripts include:

* remove-script
* add-script
* list-script
* remove-plan
* add-plan
* list-plan
* remove-goal
* add-goal
* get-value
* put-value
* max-value
* min-value

10) When the PA project reaches the simulation phase, it may prove necessary that the aircraft parameter values, including their ranges and defaults, will vary based on whether the aircraft is in the cruise, ingress, or egress modes. For example, the typical airspeed for ingress and egress is much higher than while cruising along.

11) Standard functions that the pilot would request could be predefined in the editor. For example, the pilot may wish to:

* compute fuel usage to some destination
* compute time of arrival (TOA)

12) When the simulation phase is reached, a list of antonym items may be considered for assistance in specifying allowable side-effects, since these antonyms will represent mutually exclusive aircraft states. For example, the pilot cannot be in CCIP mode and CCRP mode at the same time, so if he enters one, he must specify exit of the other script. The settings on the switches, dials, and buttons represent another type of mutual exclusion. The pilot cannot set a switch in two different positions simultaneously. The pilot should be able to modify these antonym items through the editor interface.

13) During simulation, if the pilot wishes to activate or deactive a currently defined script, goal, or plan he will need commands such as:

* LOAD
  * SCRIPT
  * PLAN
  * GOAL

* UNLOAD
  * SCRIPT
  * PLAN
  * GOAL

Many recommendations for enhancing the knowledge editor developed in this thesis deal with the simulation time frame, and as such are not implementable for a year or two. In the immediate future, recommendations 1) through 5) should be the primary research initiatives.

In the future, if expansion of the knowledge editor is to be continued at the Flight Dynamics Laboratory, it must

be modified to work with KEE's new release, Version 3.0, as specified in Teknowledge's new release documentation. The complexity of this retargeting is unknown by the author. Version 2.1 no longer exists on the Laboratory's Symbolics 3670 LISP Machine.

## Appendix A: CCRP Frame Representation Results

An example of the computer's internal frame representation of the CCRP script, originally developed in Chapter III, based on additional input from the domain expert Major Frank is as follows:

```
(script (CCRP))
(constraint NIL)
(action (NIL))
(object (panel NIL) (button NIL))
(location (from NIL) (to NIL))
(feedback (NIL))
(actor (NIL))
(side-effect NIL)

(subunit (CCRP-1))
(constraint NIL)
(action (DEPRESS))
(object (panel ICP) (button A-G-MODE-BUTTON))
(location (from DESELECT) (to SELECT))
(feedback (CHECK))
(actor (PILOT))
(side-effect NIL)

(subunit (CCRP-10))
(constraint NIL)
(action (SELECT))
(object (panel MFD) (button WPN-OSB))
(location (from DESELECT) (to SELECT))
(feedback (CONFIGURE))
(actor (PILOT))
(side-effect NIL)

(subunit (CCRP-2))
(constraint NIL)
(action (SELECT))
(object (panel MFD) (button AG-SUBMODE-OSB))
(location (from NIL) (to CCRP))
(feedback (VERIFY))
(actor (PILOT))
(side-effect NIL)

(subunit (CCRP-3))
(constraint NIL)
(action (SELECT))
(object (panel MFD) (button FCR-OSB))
(location (from DESELECT) (to SELECT))
(feedback (CONFIGURE))
```

```
(actor (COMPUTER))
(side-effect NIL)

(subunit (CCRP-8))
(constraint NIL)
(action (SELECT))
(object (panel MFD) (button RADAR-MODE-OSB))
(location (from NIL) (to GM))
(feedback (NIL))
(actor (COMPUTER))
(side-effect NIL)

(subunit (CCRP-9))
(constraint NIL)
(action (SET))
(object (panel MFD) (button TGP-OSB))
(location (from DESELECT) (to SELECT))
(feedback (VERIFY))
(actor (PILOT))
(side-effect NIL)

(subunit (CCRP-4))
(constraint NIL)
(action (SELECT))
(object (panel MFD) (button AIMING-OSB))
(location (from TGT) (to OA1))
(feedback (VERIFY))
(actor (PILOT))
(side-effect NIL)

(subunit (CCRP-5))
(constraint NIL)
(action (SELECT))
(object (panel ICP) (button STEERPT-ROCKER-SWITCH))
(location (from SP-5) (to SP-7))
(feedback (ADVISE))
(actor (COMPUTER))
(side-effect NIL)

(subunit (CCRP-11))
(constraint NIL)
(action (SET))
(object (panel LEFT-MISC) (button MASTER-ARM-SWITCH))
(location (from OFF) (to ARM))
(feedback (VERIFY))
(actor (PILOT))
(side-effect NIL)

(subunit (CCRP-6))
(constraint NIL)
(action (DEPRESS))
(object (panel HANDS-ON) (button WEAPON-RELEASE-BUTTON))
(location (from RELEASE) (to DEPRESS))
(feedback (VERIFY))
```

```
(actor (PILOT))
(side-effect NIL)

(subunit (CCRP-7))
(constraint ((IF RIPPLE-RELEASE = TRUE)
             (THEN WEAPON-RELEASE-BUTTON = HOLD)
             (UNTIL NUMBER-OF-SELECTED-STORES-TYPE = 0)))
(action (RELEASE))
(object (panel HANDS-ON) (button WEAPON-RELEASE-BUTTON))
(location (from NIL) (to RELEASE))
(feedback (VERIFY))
(actor (PILOT))
(side-effect (MASTER-ARM-SWITCH = OFF))
```

## Appendix B: CCRP LISP Generation Results

The output of the LISP code generator module using the CCRP-script's internal frame representation as shown in Appendix A appears as:

```
(defun CCRP ()
  (send lisp.listener :clear-window)
  (system-sensor-check (quote nil) (quote nil) (quote nil)
    (quote nil) (quote nil) (quote nil) (quote nil))
  (send lisp.listener :fresh-line)
  (system-sensor-check (quote check) (quote pilot) (quote
    icp) (quote a-g-mode-button) (quote depress) (quote
    deselect) (quote select))
  (send lisp.listener :fresh-line)
  (system-sensor-check (quote configure) (quote pilot)
    (quote mfd) (quote wpn-osb) (quote select) (quote
    deselect) (quote select))
  (send lisp.listener :fresh-line)
  (system-sensor-check (quote verify) (quote pilot) (quote
    mfd) (quote ag-submode-osb) (quote select) (quote nil)
    (quote ccrp))
  (send lisp.listener :fresh-line)
  (system-sensor-check (quote configure) (quote computer)
    (quote mfd) (quote fcr-osb) (quote select) (quote
    deselect) (quote select))
  (send lisp.listener :fresh-line)
  (system-sensor-check (quote nil) (quote computer) (quote
    mfd) (quote radar-mode-osb) (quote select) (quote nil)
    (quote gm))
  (send lisp.listener :fresh-line)
  (system-sensor-check (quote verify) (quote pilot) (quote
     mfd) (quote tgp-osb) (quote set) (quote deselect)
    (quote select))
  (send lisp.listener :fresh-line)
  (system-sensor-check (quote verify) (quote pilot) (quote
    mfd) (quote aiming-osb) (quote select) (quote tgt)
    (quote oal))
  (send lisp.listener :fresh-line)
  (system-sensor-check (quote advise) (quote computer)
    (quote icp) (quote steerpt-rocker-switch) (quote select)
    (quote sp-5) (quote sp-7))
  (send lisp.listener :fresh-line)
  (system-sensor-check (quote verify) (quote pilot) (quote
    left-misc) (quote master-arm-switch) (quote set) (quote
    off) (quote arm))
  (send lisp.listener :fresh-line)
  (system-sensor-check (quote verify) (quote pilot) (quote
    hands-on) (quote weapon-release-button) (quote depress)
    (quote release) (quote depress))
```

```
    (send lisp.listener :fresh-line)
    (cond ((equal ripple-release (quote t))
           (loop until (equal number-of-selected-stores-type
                              (quote 0)))
          (setq weapon-release-button (quote hold))))
    (system-sensor-check (quote verify) (quote pilot) (quote
      hands-on) (quote weapon-release-button) (quote release)
      (quote nil) (quote release))
    (send lisp.listener :fresh-line)
    (setq master-arm-switch (quote off))
)
```

## Bibliography


1. Abrett, Glenn and Mark H. Burstein. "The BBN Laboratories Knowledge Acquisition Project: KREME Knowledge Editing Environment," Proceedings: Expert Systems Workshop, 1-21 (April 1986).

2. Air Force Wright Aeronautical Laboratories, Aeronautical Systems Divsion, Air Force Systems Command. Statement of Work F33615-85-C-3804. Wright-Patterson AFB, OH, 22 May 1985.

3. Barr, Avron and Edward A. Feigenbaum, eds. The Handbook of Artificial Intelligence, Volume 1. Los Altos CA: William Kaufmann, Inc., 1981.

4. Barr, Avron and Edward A. Feigenbaum, eds. The Handbook of Artificial Intelligence, Volume 2. Los Altos CA: William Kaufmann, Inc., 1982.

5. Buchanan, Bruce G. and Edward H. Shortliffe. Rule-Based Expert Systems. Reading MA: Addison-Wesley Publishing Company, 1984.

6. Capozzi, Maj Rocky, USAF F-16 pilot, stationed in Germany. Personal interview, AFIT, Wright-Patterson AFB, OH, July 1986.

7. Cohen, Paul R. and Edward A. Feigenbaum, eds. The Handbook of Artificial Intelligence, Volume 3. Los Altos CA: William Kaufmann, Inc., 1982.

8. Cross, Stephen E., "Expert System Requirements for Flight Domain Applications," Conference on Intelligent Systems and Machines, 220-224 (24-25 April 1984).

9. Cross, Maj Stephen E., AFWAL Deputy for Advanced Computer Technology, Air Force Institute of Technology. Personal interview, AFIT, Wright-Patterson AFB, OH, March-November 1986.

10. Cross, Capt Stephen E., "Intelligent Avionics - AI Technology Insertion Opportunities," Third Annual Conference on Applied Artificial Intelligence, 49-59 (March 1985).

11. Cross, Stephen E. et al. "Knowledge-Based Pilot Aids: A Case Study in Mission Planning," Artificial Intelligence and Man-Machine Systems, Bonn, Germany, 141-174 (May 1986).

12. Department of the Air Force. Nonnuclear Munitions Delivery, USAF Series F-16C and F-16D Aircraft. T.O. 1F-16C-34-1-1. Fort Worth Division: General Dynamics, 4 March 1985.

13. Fairley, Richard E. Software Engineering Concepts. New York: McGraw-Hill Book Company, 1985.

14. Faletti, Joseph, "PANDORA - A Program for Doing Commonsense Planning in Complex Situations," 1985 IEEE Region 5 Conference, 185-188 (13-15 March 1985).

15. Frank, Maj Richard, USAF F-16 pilot, F-16 System Program Office. Personal interview, ASD, Wright-Patterson AFB, OH, July-November 1986.

16. Geddes, Norman D., "Intent Inferencing Using Scripts and Plans", Proceedings of the First Annual Aerospace Applications of Artificial Intelligence Conference, 160-172 (16-19 September 1985).

17. Geddes, Norman D., PhD Candidate, The Georgia Institute of Technology. Personal interview, Georgia Institute of Technology, Atlanta, GA, 24-25 March 1986.

18. Geddes, Norman D., PhD Candidate, The Georgia Institute of Technology. Personal interview, Air Force Wright Aeronautical Laboratories, Flight Dynamics Laboratory, Wright-Patterson AFB, OH, 15 October 1986.

19. General Dynamics. F-16C/D Avionic System Manual (Block 25B). 16PR3927; Volume 2; Revision A. Fort Worth Division, 15 May 1985.

20. Harandi, Mehdi T., "A Knowledge-Based Programming Support Tool," IEEE Proceedings: Trends and Applications, 233-239 (25-26 May 1983).

21. Howatt, Capt James W., Lecture notes in EENG 593, Software Engineering. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, March-June 1986.

22. Kelly, Dr Clint., Director of DARPA's Strategic Computer Program. Address to AFIT students in the Artificial Intelligence sequence. Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, 14 October 1986.

23. Knode, Capt David L. An Approach to Planning in the Inflight Emergency Domain, MS Thesis, GCS/ENG/84D-15. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, December 1984.

24. McGraw, Bruce A. and Karen L. McGraw, "Military Applications of Artificial Intelligence Technology," IEEE Region 5 Conference, 31-34 (13-15 March 1985).

25. Morishige, Lt Col Ronald I. and Lt Col John P. Retelle, Jr., "Air Combat and Artificial Intelligence," Air Force Magazine, 91-93 (October 1985).

26. Morishige, Lt Col Ronald I., "The Pilot's Associate Program," Pilot's Associate Office Information File prepared for News Release, ASD Public Affairs, Wright-Patterson, AFB, OH, 1-9 (January 1986).

27. Rich, Elaine. Artificial Intelligence. New York: McGraw-Hill Book Company, 1983.

28. Schank, Roger C. and Christopher K. Riesbeck, eds. Inside Computer Understanding. Hillsdale NJ: Lawrence Erlbaum Associates, Publishers, 1981.

29. Schank, Roger C. and R. Abelson. Scripts, Plans, Goals, and Understanding. Hillsdale NJ: Lawrence Erlbaum Associates, Publishers, 1977.

30. Shapiro, Daniel G. and Brian P. McCune, "The Intelligent Program Editor," IEEE Proceedings: Trends and Applications, 226-232 (25-26 May 1983).

31. Stein, Kenneth J., "DARPA Stressing Development of Pilot's Associate System," Aviation Week & Space Technology, 69-74 (22 April 1985).

32. Stockbridge, Capt Samuel E. Autonomous Vehicle Mission Planning Using AI Techniques, MS Thesis, GE/ENG/85D. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, December 1985.

33. Teknowledge, Inc. Knowledge Engineering Methodology. Palo Alto CA: Teknowledge, Inc., 1985.

34. Van der Weert, Maj Ron, USAF F-15 pilot, Air Force Wright Aeronautical Laboratories, Pilot Associate Program Office. Personal interview, AFWAL, Wright-Patterson AFB, OH, October 1986.

35. Waters, Richard C., "KBEmacs: Where's the AI?," The AI Magazine, 47-56 (Spring 1986).

36. Webster's New World Dictionary (Second College Edition), edited by David B. Guralnik et al. New York: William Collins and World Publishing Co., Inc., 1978.

37. Wilensky, Robert, "Meta-Planning : Representing and Using Knowledge About Planning in Problem Solving and Natural Language Understanding," Cognitive Science, 5: 197-233 (1981).

38. Wilensky, Robert. Planning and Understanding. Reading MA: Addison-Wesley Publishing Co., 1983

39. Wilensky, Robert, "Talking to UNIX in English: An Overview of an On-line UNIX Consultant," The AI Magazine, 29-39 (Spring 1984).

40. Winston, Patrick Henry. Artificial Intelligence. Reading MA: Addison-Wesley Publishing Company, 1977.

41. "1986 Flight Mishap Forecast," Flying Safety, 42: 6-8 (February 1986).

# VITA

Captain Darleen Avery Sobota was born on 1 April 1960 at Hahn Air Base, Germany. She graduated from high school in Panama City, Florida, in 1978 and attended the United States Air Force Academy from which she received the degree of Bachelor of Science in Computer Science in June 1982. Upon graduation, she was commissioned in the Regular Air Force, and reported to Wright-Patterson Air Force Base where she served as a computer engineer in the Flight Dynamics Laboratory. While working with future cockpit display technology in simulators, she took classes part-time in electrical engineering from the Air Force Institute of Technology. After three years in the Laboratory, she entered the Air Force Institute of Technology, School of Engineering, in May 1985 for a Master of Science in Computer Engineering and has pursued the Artificial Intelligence sequence. Concurrently, she was awarded a second Bachelor of Science degree, this time in Electrical Engineering in December 1985.

Permanent address: 623 Barton Avenue
Panama City, FL 32404

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

AD-A177 [illegible]

## REPORT DOCUMENTATION PAGE

| | |
|---|---|
| 1a REPORT SECURITY CLASSIFICATION: UNCLASSIFIED | 1b RESTRICTIVE MARKINGS: |
| 2a SECURITY CLASSIFICATION AUTHORITY: | 3 DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited. |
| 2b DECLASSIFICATION/DOWNGRADING SCHEDULE: | |
| 4 PERFORMING ORGANIZATION REPORT NUMBER(S): AFIT/GCE/ENG/86D-4 | 5 MONITORING ORGANIZATION REPORT NUMBER(S): |
| 6a NAME OF PERFORMING ORGANIZATION: School of Engineering; 6b OFFICE SYMBOL (If applicable): AFIT/EN | 7a NAME OF MONITORING ORGANIZATION: |
| 6c ADDRESS (City, State and ZIP Code): Air Force Institute of Technology, Wright-Patterson AFB, Ohio 45433 | 7b ADDRESS (City, State and ZIP Code): |
| 8a NAME OF FUNDING/SPONSORING ORGANIZATION: Pilot's Associate Office; 8b OFFICE SYMBOL (If applicable): AFWAL/CCU | 9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: |
| 8c ADDRESS (City, State and ZIP Code): Wright-Patterson AFB, Ohio 45433 | 10 SOURCE OF FUNDING NOS: PROGRAM ELEMENT NO; PROJECT NO; TASK NO; WORK UNIT NO |
| 11 TITLE (Include Security Classification): See Box 19 | |
| 12 PERSONAL AUTHOR(S): Darleen Avery Sobota, Captain, USAF | |
| 13a TYPE OF REPORT: MS Thesis; 13b TIME COVERED: FROM _____ TO _____; 14 DATE OF REPORT (Yr, Mo, Day): 1986 December; 15 PAGE COUNT: 150 | |
| 16 SUPPLEMENTARY NOTATION: | |

| 17 COSATI CODES | | | 18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number) |
|---|---|---|---|
| FIELD | GROUP | SUB GR | |
| 06 | 04 | | Artificial Intelligence, Planning, Knowledge Representation, Pilot's Associate Knowledge Acquisition, Script-Based Planning |
| 09 | 02 | | |

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

Title: PROTOTYPE KNOWLEDGE ACQUISITION AND REPRESENTATION EDITOR FOR THE F-16 FLIGHT DOMAIN

Thesis Advisor: Stephen E. Cross, Major, USAF

#18: Editor, Prototype

Approved for public release: IAW AFR 190-1.
LYNN E. WOLAVER
Dean for Research and Professional Development
Air Force Institute of Technology
Wright-Patterson AFB OH 45433

| | |
|---|---|
| DISTRIBUTION/AVAILABILITY OF ABSTRACT: UNCLASSIFIED/UNLIMITED X SAME AS RPT ☐ DTIC USERS ☐ | 21 ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED |
| 22a NAME OF RESPONSIBLE INDIVIDUAL: Stephen E. Cross, Major, USAF | 22b TELEPHONE NUMBER (Include Area Code): (513) 255-3576; 22c OFFICE SYMBOL: AFIT/ENG |

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

## Abstract

Current research in the study of intent inferencing for the fighter aircraft flight domain is hampered by the problem, "How should knowledge required about the flight domain be acquired and represented?" Without automated tools for acquiring and representing knowledge, large expert system development will continue to be hindered.

The purpose of this research was the initial design and construction of a prototype editor to assist in the acquisition and representation of script-based knowledge in the context of the F-16C flight domain. The editor was responsible for: maintaining the internal consistency of the knowledge base when knowledge was added, updated, or deleted; preventing syntax errors in formatting by providing cascading menus for domain specific knowledge entry, while leaving the frame-based formatting details to the editor tool, and; preventing spelling errors among already existing (created) knowledge items by providing pilot selectable pop-up menus reflecting the current state of valid choices in the knowledge base for a particular item. Other potential benefits of an automated knowledge acquisition and representation editor are discussed.

As designed and implemented, the editor generates an internal frame-based representation of the pilot's air-to-ground weapon delivery checklist information, and further, generates a LISP computer program from this internal representation, without requiring the pilot to know the computer language.

The prototype editor was implemented on a Symbolics 3670 LISP machine, in LISP, supported by Intellicorp's Knowledge Engineering Environment (KEE) frame-based expert system building tool. In the future, integrating plan-based knowledge into the knowledge editor tool should allow testing of the concept of intent inferencing, supporting the work of the Pilot's Associate Office.

END

4-87

DTIC